Universidad Zaragoza
1542

Trabajo Fin de Grado

Casos de uso de la estandarización de una plataforma de servicios de automoción para OSVehicle

Use case of the standardization of an automotive service platform over OSVehicle.

Autor

Alejandro Hernández Pardos

Director

Francisco Javier Solans Benedí

Ponente

Francisco Javier Zarazaga Soria

Escuela de Ingeniería y Arquitectura
2016/2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

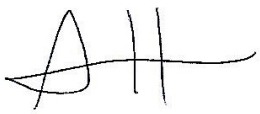
D./Dª. ALEJANDRO HERNÁNDEZ PARDOS,

con nº de DNI 73014985-J en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) GRADO, (Título del Trabajo) CASOS DE USO DE LA ESTANDARIZACIÓN DE UNA PLATAFORMA DE SERVICIOS DE AUTOMOCIÓN PARA OSVEHICLE,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 23-NOVIEMBRE-2017



Fdo: ALEJANDRO HERNÁNDEZ PARDOS

Agradecimientos

A Paco por darme la oportunidad de realizar este trabajo y por toda la ayuda y conocimientos que me ha proporcionado.

A los compañeros de Neodoo por los buenos ratos pasados.

A Francisco Javier Zarazaga por su tutelación y sus consejos.

A mis compañeros de la carrera, con los que he compartido tantos buenos, y malos momentos.

A mis amigos, que siempre me han apoyado y me han animado cuando lo he necesitado.

Y sobre todo a mi familia, que siempre han confiado en mi y me han animado a continuar en los peores momentos. Si no fuera por ellos no habría llegado nunca tan lejos.

Casos de uso de la estandarización de una plataforma de servicios de automoción para OSVehicle

RESUMEN

En la actualidad cualquier persona puede tener un smartphone, debido a ello se ha producido un increíble auge de las aplicaciones disponibles para satisfacer las múltiples necesidades de los usuarios, que van desde aplicaciones que monitorizan el sueño hasta aplicaciones que te indican cuando ir al baño en el cine para no perderte nada interesante. Este hecho junto al de prácticamente todos los vehículos modernos, especialmente en el caso de los vehículos eléctricos, tienen control por software de los elementos del mismo han hecho posible el desarrollo de este trabajo fin de grado, ya que combina ambas características.

El presente trabajo fin de grado ha consistido en la elaboración de una prueba de concepto de cara a abrir nuevas líneas de negocio para la empresa. Para ello se han elaborado versiones preliminares de tres piezas básicas: creación de una aplicación móvil que permita controlar un vehículo dentro de la aproximación de OSVehicle con el teléfono móvil; un simulador de un vehículo en el que se ha utilizado una *RapsBerry* representando al ordenador a bordo; y una API de servicios expuestos por el vehículo para su consumo por la aplicación móvil. En este punto se ha hecho una apuesta por una API REST que ofrece las funcionalidades del vehículo y del protocolo que la comunica con el ordenador del coche. Esta API REST está basada en la que usa Tesla para comunicarse con sus vehículos. En el caso de la aplicación móvil, se ha buscado un escenario demostrador de posibilidades y con este fin se ha construido una aplicación móvil para Android y un servicio web para simular un negocio de carsharing (servicio de alquiler de vehículos por hora). Esto ha cumplido el doble papel de actuar de ejemplo demostrado y posibilitar la comprobación del correcto funcionamiento del sistema. Además también se ha automatizado el despliegue de todo el sistema.

El resultado ha sido el desarrollo de una capa de acceso a un vehículo simulado, así como la creación de una aplicación Android básica que nos permite utilizarla para simular un negocio de alquiler de vehículos por minutos, consiguiendo asentar las bases para poder implementar la parte electrónica del vehículo y así poder cumplir el objetivo de controlar el vehículo con el móvil en un escenario real.

Índice general

Índice de tablas

Índice de ilustraciones

Capítulo 1

Introducción

Este capítulo expone brevemente los aspectos más importantes del Trabajo Fin de Grado, en adelante TFG, para obtener una visión global del trabajo realizado.

La finalidad de este TFG ha sido desarrollar un sistema que nos permita obtener información y controlar un vehículo remotamente. Para ello se ha emulado la API de Tesla y como demostrador funcional de esta API se ha simulado un sistema de alquiler de vehículos a través del teléfono móvil.

1.1 Contexto

El TFG ha sido realizado en la empresa Neodoo Microsystems S.L. bajo la dirección de Francisco Javier Solans Benedí, CEO de la misma. El ponente ha sido Francisco Javier Zarazaga Soria, Profesor Titular del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza.

Neodoo Microsystems S.L. es una empresa con mas de diez años de experiencia en el Sector de las Tecnologías y la información bajo el paradigma del Software Libre[1]. Neodoo Microsystems ofrece servicios TIC (Tecnologías de la información y comunicaciones) en un marco de calidad basado en código libre y estándares abiertos.

Son expertos en Middleware, soluciones linux, Apache, Jboss, aquitecturas de alta disponibilidad y desarrollo front-end.

Perteneciente a CESLA[1] y ASOLIF [2] siempre ha estado interesado en proyectos Open Source internacionales como OSVehicle[2]. Este proyecto está explicado detalladamente en el Anexo A.

1.2 Motivación

En pleno siglo XXI con la crisis del cambio climático y el auge de las energías renovables, los vehículos eléctricos han ido ganando terreno frente a los vehículos de motor de combustión. Cada vez son más marcas las que sacan un modelo eléctrico, y cada vez se implantan mas zonas donde poder cargarlos (En nuestra ciudad, Zaragoza, hay 36 puntos de carga disponibles). Por este motivo nace el proyecto Open Source denominado OSVehicle en 2012, que desarrolla este concepto para que pueda ser accesible por todos nosotros. Este proyecto te permite en menos de una hora y por una cantidad entre los 4.000 y 6.000 euros montar tu propio OSVehicle con motor híbrido o totalmente eléctrico. Así nace también *TABBY* que es una plataforma Open Source para vehículos de OSVehicle. *Tabby* es la primera estructura Open Source para la creación de vehículos. Se le ha definido como “Ikea Car” y el “Arduino para la movilidad”. Se trata de una plataforma

1 Clúster de Entidades pro Software Libre de Aragón. (http://www.cesla.info/)
2 Federación Nacional de empresas de Software libre. (http://www.asolif.es)

versátil que puede ser usada para hacer tu propio vehículo, para fines educativos y muchos más.

Un elemento característico de prácticamente todos los vehículos modernos, especialmente en el caso de los vehículos eléctricos, es el control por software de todos los elementos del mismo. Esto ha abierto la puerta al desarrollo de sistemas externos que desde un móvil o una aplicación Web pueden efectuar operaciones sobre los mismos. En este sentido, el primer planteamiento que se propuso fue el desarrollo de una capa software sobre OSVehicle que facilitase este tipo de construcción de sistemas de información externos al vehículo. Aunque muy prometedora, se vio enseguida (incluso antes de elaborar la propuesta de TFG) que este trabajo sobrepasaba las posibilidades disponibles por la empresa (en lo referente a costes de infraestructura) y el alcance del TFG. No obstante, se hizo una apuesta por ir dando pasos en esta línea para lo que se buscó un alcance y compromiso de recursos viable. Es en este punto se optó por hacer una pequeña simulación del OSVehicle mediante un miniordenador (se optó por una *Rapsberry[3]*) y construir una capa de acceso tomando como base la API que Tesla tiene para el control de sus vehículos.

Tesla, Inc.[4] (anteriormente llamada Tesla Motors) es una compañía estadounidense ubicada en Silicon Valley, California, que diseña, fabrica y vende coches eléctricos, componentes para la propulsión de vehículos eléctricos y sistemas de almacenamiento a baterías. Tesla se fundó en 2003 por un grupo de ingenieros que querían probar que la gente no tenía que realizar concesiones para conducir vehículos eléctricos, y que estos podían ser mejores, más rápidos y más divertidos de conducir que los vehículos de gasolina. A día de hoy, Tesla se ha convertido en el referente mundial en vehículo eléctrico. Sus vehículos están completamente controlados por sistemas software como paso indispensable para ser operados como vehículos autónomos. En esta línea, Tesla ofrece diferentes niveles de control ofertados al resto de sistemas mediante API con diferentes niveles de acceso. Entre estos, Tesla cuenta con una especificación de una API para que sea posible el desarrollo de sistemas externos al vehículo. El objetivo inicial de la misma ha sido la especificación de la comunicación con las aplicaciones de terceros como VisibleTesla, RemoteS que la usan para controlar las funciones del vehículo o para acceder a los datos proporcionados por Tesla. Es esta API la que se ha decidido emular en el contexto de este TFG. Esta API móvil no está oficialmente documentada por Tesla y no está destinada a ser utilizada por nadie más que por los empleados de la compañía. Entre otras cosas permite arrancar el vehículo, abrir las puertas y tocar el claxon. Pero, al igual que muchas cosas en estos días, si existe puede ser resuelto, y algunos propietarios de vehículos Tesla enérgicos han descubierto cómo funciona. Hay un sitio gratuito llamado Apiary[5] que se puede usar para documentar las API y que se usó para documentar la API de Tesla Model S.

En el Anexo B. se encuentra detallada la API de Tesla según se ofrece en el mencionado sitio Web.

Aunque finalmente no se ha podido realizar la capa de software para los OSVehicles, que hubiera sido lo ideal para la empresa, con este TFG se trabaja en la línea deseada por la misma ya que se produce un acercamiento al proyecto de OSVehicle y se simula la API de Tesla que actualmente es un referente en cuanto a los vehículos eléctricos. Esto permite que la empresa pueda comenzar a manejar tecnologías y propuestas de base para la creación de una nueva línea de negocio. Además, NEODOO tiene interés en la parte correspondiente a la autenticación mediante el servidor *Keycloak[6]* para futuros proyectos así como de la parte relacionada con el servidor *MQTT[7]*. *MQTT* (Message Queue Telemetry Transport) es un protocolo de mensajería basada en publicación-suscripción estándar ISO (ICE / IEC PRF 20922). Funciona sobre el protocolo TCP/IP y está diseñado para conexiones con ubicaciones remotas donde se requiere una "huella de código pequeña" o el ancho de banda de la red es limitado. El patrón de publicación publicación-suscripción requiere un intermediario de mensajes, o broker.

1.3 Objetivos

Los objetivos planteados para este TFG son:

1. Realizar un estudio previo de la API de Tesla.
2. Creación de nuestra API REST que simula la de Tesla.
3. Emular la comunicación de la API de Tesla con los vehículos (Se usará un servidor *MQTT* y una *raspBerry* para emular el vehículo).
4. Desarrollo de un servicio web que utilice nuestra API REST. Se ha elegido un escenario de alquiler de coches donde mediante una aplicación móvil se pueden alquilar vehículos
5. Desarrollo de la aplicación móvil.
6. Garantizar la privacidad mediante un control de acceso con autenticación.
7. Por ultimo se procederá a la automatización del sistema.

1.5 Organización de la memoria

El presente documento se encuentra estructurado de la siguiente manera:

- En el **capítulo 1**, se ha realizado una breve introducción al trabajo realizado.
- En el **capítulo 2,** se describe todo el trabajo realizado durante el proyecto.
- En el **capítulo 3,** se describe la valoración del proyecto, según los resultados obtenidos y la económica.
- En el **capítulo 4,** se describen las conclusiones del proyecto, y las posibles lineas de trabajo futuro.

En cuanto al apartado de anexos se encuentra de la siguiente manera:

- En el **Anexo A,** se describe en que consiste el proyecto OSVehicle.
- En el **Anexo B,** se describe todas las funcionalidades que ofrece la API de Tesla.
- En el **Anexo C,** se describe en detalle la fase de análisis.
- En el **Anexo D,** se describe en detalle la fase de diseño.
- En el **Anexo E,** se describe en detalle la fase de validación.
- En el **Anexo F**, se describe el manual de configuración.
- En el **Anexo G**, se describe el manual de usuario.

Capitulo 2.

Trabajo realizado.

En este capítulo se verán las diferentes fases del desarrollo de este trabajo con vistas a la consecución de los objetivos expuestos en el capítulo anterior.

2.1 Visión Global del trabajo.

Esta sección muestra de manera conjunta todos los elementos que integran el sistema construido. Aquí aparecen tanto desarrollos específicos del proyecto, como sistemas ya existentes (Open Source) que se han integrado (y para los que en muchos casos ha sido necesaria llevar a cabo una parametrización).

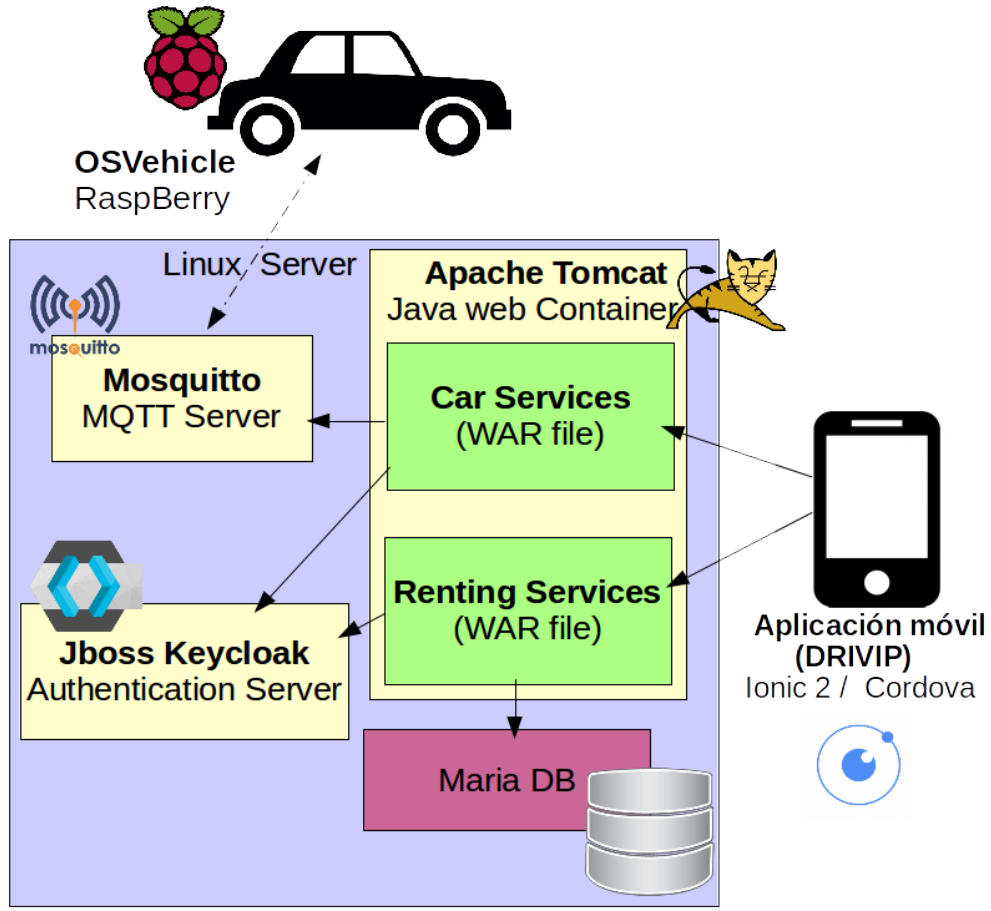


Ilustración 1: Esquema general del sistema.

En la figura 1 se pueden identificar fácilmente los componentes de nuestro sistema. Estos son:

- **Car Services**: Este componente emula la API de Tesla. Por tanto se ha desarrollado una API REST encargada de gestionar las funcionalidades de nuestro vehículo. Esta API publicará los mensajes correspondientes en nuestro Servidor MQTT para que el vehículo correspondiente realice una función determinada.
- **OSVehicle**: Este componente representa un vehículo, se ha utilizado una RapsBerry como representación del vehículo. La RapsBerry tiene un pequeño programa java denominada osvehicle-client que esta subscrito al servidor *MQTT* y mostrará por la salida estándar los mensajes que reciba. De esta manera se comprueba que el OSVehicle recibe las ordenes correctamente.
- **Renting Services**: Este componente gestiona toda la lógica de negocio del sistema de alquiler de vehículos que se ha simulado para demostrar la funcionalidad de la API. Hay que recordar que este TFG no ha perseguido construir un sistema de alquiler de vehículos sino tan solo contar con un demostrador de tecnología para probar la funcionalidad de la API simulada. Es el encargado de recibir las peticiones de la aplicación móvil mediante una API REST, el encargado de autenticar estas peticiones mediante el servidor de autenticación Keycloak. El servidor Keycloak se explica más adelante. También es el encargado de mantener la persistencia de la Base de Datos.
- **Aplicación Móvil**: Se ha desarrollado una aplicación móvil mediante el FrameWork *Ionic 2[8]* compatible con teléfonos Android. Esta aplicación simula un sistema de alquiler de coches, concretamente un servicio de *carsharing.* La aplicación se comunicará tanto con el módulo Renting Services, como con el módulo Car Services. Además solicitará el Access Token al servidor *Keycloak.*
- **Mosquitto**: Se ha elegido el servidor *Mosquitto* como broker de *MQTT*. *Mosquitto* es un servidor *MQTT* Open Source. En la sección 2.2.2. Análisis del protocolo MQTT se explica el protocolo *MQTT.* Este broker es el servidor que acepta mensajes publicados por clientes y los difunde entre los clientes suscritos.
- **JBoss Keycloak**: Se ha elegido el servidor *JBoss Keycloak* como servidor de autenticación para nuestro sistema. *Keycloak* es un proveedor de *OAuth[9] (*OAuth es un protocolo que permite autorización de una API. Esta explicado en la sección 2.2.3. Análisis del Protocolo OAUth2.) que se puede usar en las aplicaciones y servicios para proporcionar seguridad a los servicios REST.
- **Maria DB:** Se ha elegido Maria DB[10] como sistema de gestión de bases de datos. Maria DB es un proyecto Open Source derivado de Mysql. La Base de Datos contendrá la información necesaria para realizar el simulador de alquiler de vehículos.

2.2 Análisis.

Este apartado describe el análisis que ha sido llevado a cabo antes de desarrollar la aplicación para conocer la mejor manera de abordar este proyecto.

2.2.1 Análisis de la API de Tesla

Como ya se ha explicado anteriormente, si hay un referente a la hora de hablar de coches eléctricos este es sin duda Tesla.

Actualmente comercializan 3 modelos: el modelo S, el modelo X y el modelo 3. Se ha tomado como referencia la API del modelo S. Esta API nos permite monitorizar y controlar nuestro Tesla Modelo S remotamente.

Entre las funcionalidades mas destacadas de esta API están por un lado las que nos permiten conocer información sobre nuestro vehículo:

- Conocer el estado de carga de la batería.
- Conocer la temperatura interior y exterior del vehículo.
- Conocer la posición y velocidad del vehículo.
- Conocer el estado en general del vehículo.

Por el otro lado están las funcionalidades que nos permiten controlar el vehículo.

- Encender el vehículo.
- Establecer el limite de carga.
- Empezar y parar la carga de la batería.
- Abrir y cerrar las puertas,.
- Configurar la temperatura de los asientos.

En el Anexo B. se explica esta API en detalle con todas sus funcionalidades.

Para nuestro proyecto solo nos interesan algunas de estas funcionalidades:

1. Abrir puertas: Esta funcionalidad es la mas importante, ya que nos permite abrir la puerta remotamente con nuestro móvil una vez hemos seleccionado el vehículo con el que queremos viajar.
2. Cerrar puertas: Igual que abrir puertas, esta funcionalidad nos permite cerrar el coche una vez que hemos acabado nuestro trayecto.

Aunque se ha dejado dejado implementadas todas las funcionalidades, para nuestro proyecto solo necesitaremos estas dos.

2.2.2. Análisis del protocolo MQTT

MQTT (Message Queue Telemetry Transport) es un protocolo usado para la comunicación machine-to-machine (M2M) en el "Internet of Things"[3]. Este protocolo está orientado a la comunicación de sensores, debido a que consume muy poco ancho de banda y puede ser utilizado en la mayoría de los dispositivos.

La arquitectura de *MQTT* sigue una topología en estrella, con un nodo central que hace de servidor o "broker" con una capacidad de hasta 10.000 clientes. El broker es el encargado de gestionar la red y de transimitir los mensajes. Para mantener el canal activo los clientes mandan periódicamente un paquete (PINGREQ) y esperan la respuesta del broker (PINGRESP)[7].

En la figura 2 se puede ver el esquema de funcionamiento del protocolo *MQTT*. La comunicación se basa en unos "topics" (temas), que el cliente que publica el mensaje crea y los nodos que deseen recibirlo deben subscribirse a él. La comunicación puede ser de uno a uno o de uno a muchos. Este concepto es muy similar al que se emplea en colas, donde existen unos publicadores (que publican o emiten información) y unos subscriptores (que reciben dicha información) siempre que ambas partes estén subscritas a la misma cola.

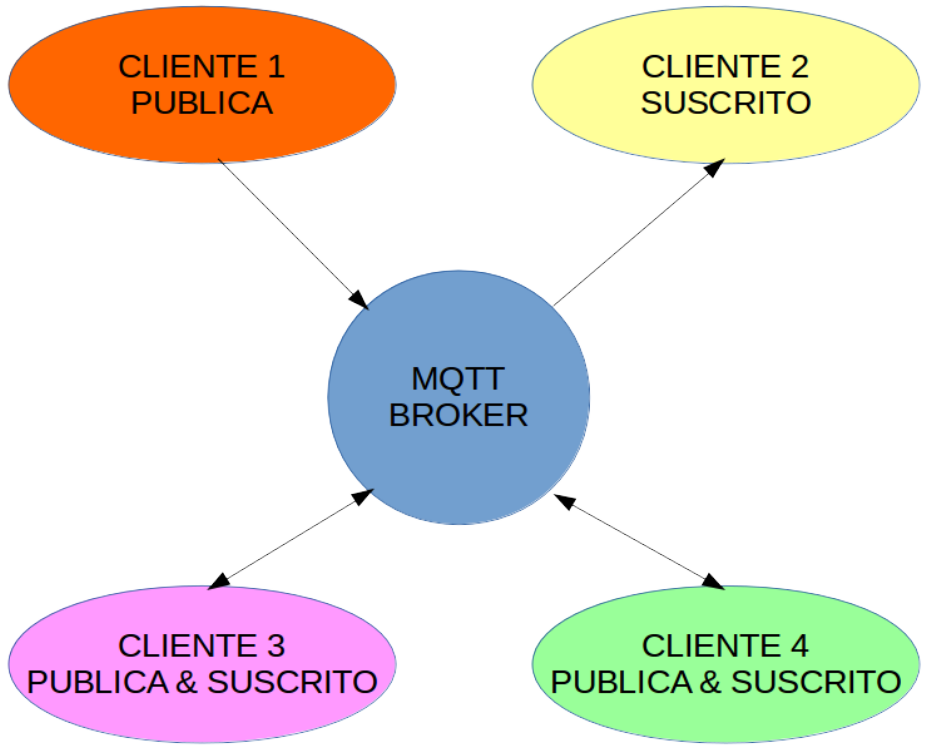


Ilustración 2: Esquema del protocolo MQTT

3 Concepto que se refiere a la interconexión digital de objetos cotidianos con Internet. (Iot)

Se ha elegido *Mosquitto* como broker. *Mosquitto* es un broker OpenSource ampliamente utilizado debido a su ligereza, lo que nos permite, emplearlo en gran número de ambientes. El protocolo *MQTT* esta más detallado en el Anexo C.1. Análisis del protocolo MQTT.

2.2.3. Análisis del Protocolo OAUth2.

OAuth2 es un protocolo usado para permitir a una aplicación acceder a los recursos de un usuario sin que este proporcione a la aplicación cliente sus credenciales y manteniendo el control de revocar los permisos concedidos.

De esta manera podremos proteger los recursos que nos interesen de manera que solo puedan acceder las peticiones que contengan en el header de su petición un Acces-Token válido.

Keycloak es la solución Open Source de autenticación y autorización centralizada de RedHat. *Keycloak* es un proveedor de OAuth que podemos usar en nuestras aplicaciones y servicios para proporcionar autenticación, autorización, SSO y también añadir seguridad a los servicios REST que desarrollemos.

Se ha elegido este proveedor de OAuth2 por su facilidad de uso y por sus múltiples adaptadores disponibles para varias plataformas y lenguajes de programación.

En la figura 3 se puede ver un esquema de la arquitectura de OAUTH2.

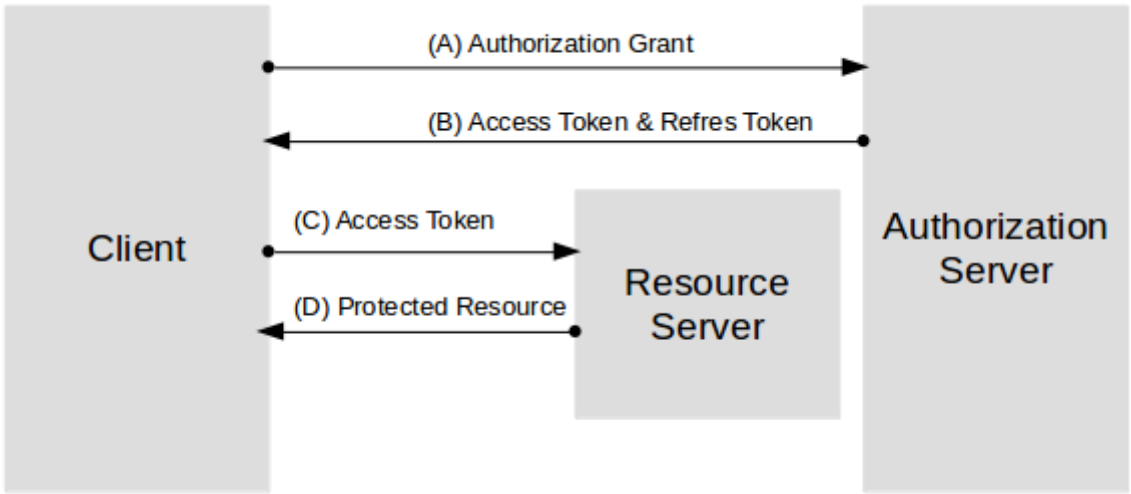


Ilustración 3: Esquema de comunicación del protocolo OAuth2.

- (A): El cliente pide un token de acceso al proveedor, presentando un Authorization Grant.
- (B): El Authorization Server, en nuestro caso *Keycloak,* valida la petición y devuelve un acces token.
- (C): El cliente pide recursos al servidor de recursos presentando el acces token.
- (D): El servidor de recursos valida el Access token y si es válido devuelve los recursos protegidos.

En el Anexo C.2. Análisis de OAUTH2. se encuentra detallada la información sobre este protocolo.

2.3. Diseño

Una vez establecidos los requisitos y realizado un análisis del sistema a desarrollar, la siguiente fase es el diseño. En este apartado se describe como se ha ido desarrollando la aplicación desde el punto de vista del diseño.

En el sistema propuesto se pueden diferenciar fácilmente las partes implicadas. En la figura 4 se puede ver el proceso que se sigue al usar nuestro sistema.

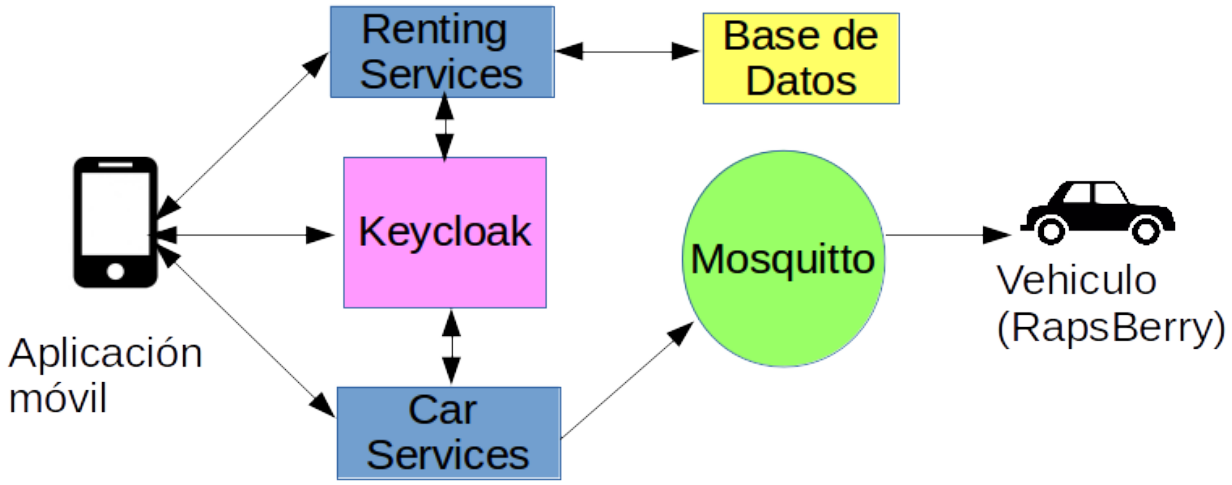


Ilustración 4: Esquema del diseño del sistema..

2.3.1. Aplicación móvil.

Como ya se ha explicado se ha desarrollado una aplicación móvil que simula un sistema de *carsharing*[4]. Esta aplicación permite al usuario registrarse, iniciar sesión y visitar su perfil. Además esta aplicación muestra los vehículos que se encuentren disponibles para alquilar, pudiendo el usuario reservar o alquilar el vehículo deseado. Cuando se encuentre

4 Vehículos compartidos o prestamos de vehículos (carsharing)

en medio de un alquiler el usuario tiene la opción de finalizar el alquiler. La aplicación móvil se comunicará con nuestro servidor de autenticación para recibir el Acces Token. Una vez tenga el Access Token se comunicará primero con el módulo Renting Services para realizar las operaciones relacionadas con el alquiler de coches y si la respuesta es satisfactoria y se desea comenzar un alquiler, la aplicación se comunicará con el módulo Car Services para abrir las puertas del vehículo deseado.

En la sección C.4. Requisitos de la aplicación móvil. se incluyen los requisitos de la aplicación móvil.

2.2.2. Renting Services.

En la figura 5 se puede ver el diseño del componente Renting Services.

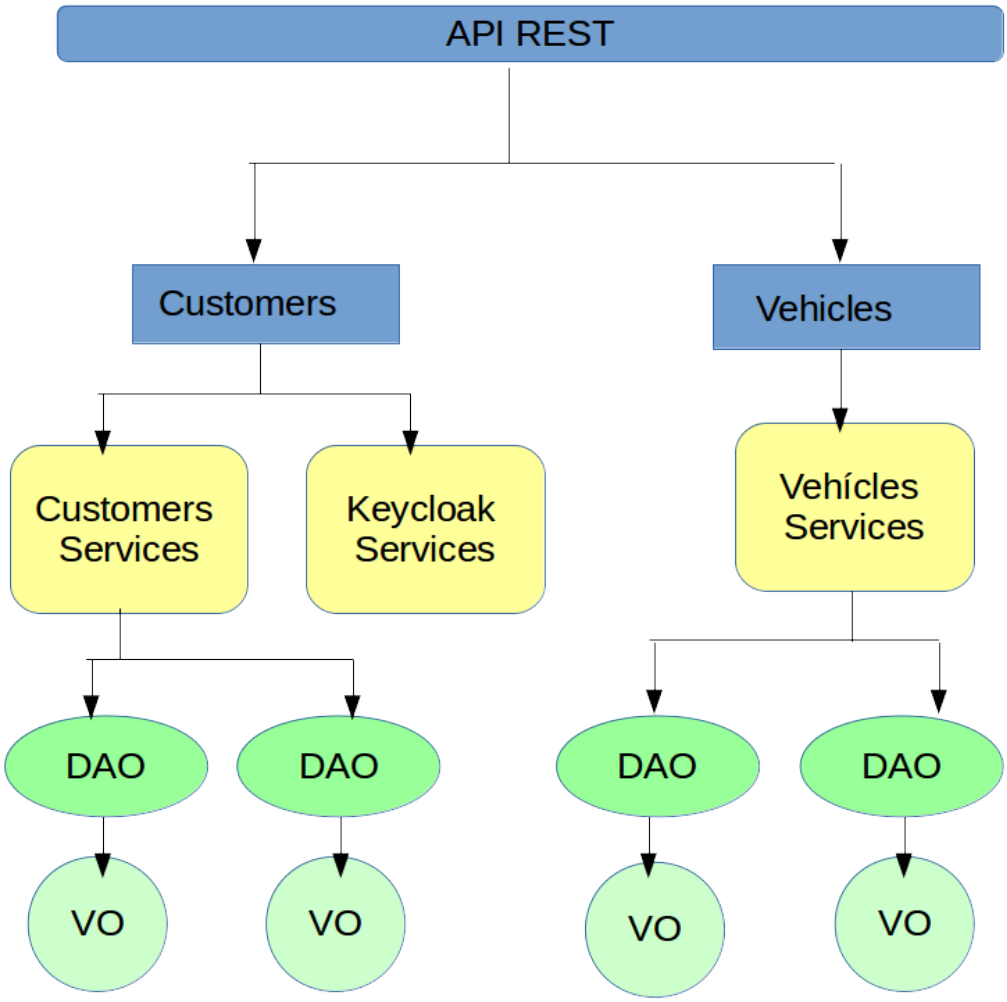


Ilustración 5: Diseño del módulo Renting Services.

Como ya se ha explicado este componente se comunica con la aplicación móvil a través de una API REST. Se pueden diferenciar dos grandes módulos:

- **Customers**: recibe las peticiones web relacionadas con los clientes, obtiene los atributos si los tiene y se los manda al siguiente elemento, en este caso Customers Services o Keycloak Services.
- **Vehicles**: recibe las peticiones web relacionadas con los vehículos, obtiene los atributos si los tiene y se los manda al siguiente elemento, en este caso Vehicles Services.

A continuación se encuentran los módulos de Services. Estos módulos son los encargados de la lógica de negocio, se encargan de transformar los datos y comunicarse con los demás módulos. Se pueden diferenciar tres módulos de servicios:

- **Customers Services**: Gestiona todo lo relacionado con los clientes. Se comunica con los DAO[5] de clientes.
- **Keycloak Services**: Gestiona la creación de usuarios en el servidor Keycloak.
- **Vehicles Services**: Gestiona todo lo relacionado con los vehículos. Se comunica con los DAO de vehículos.

Por ultimo se encuentran los módulos DAO y VO[6]. Estos módulos encapsulan el acceso a la base de datos. Por lo que cuando la capa de lógica de negocio (Módulos Services) necesite interactuar con la base de datos lo hará a través de los DAO. En una base de datos relacional hay un DAO por cada entidad. Finalmente los VO son utilizados por DAO para transportar los datos desde la Base de Datos hacia la capa de lógica de negocio y viceversa. Podría decirse que un VO es un objeto común y corriente, que tiene como atributos los datos del modelo, con sus correspondientes accessors (getters y setters).

En el Anexo D.1. Diseño de la Base de Datos se encuentra información detallada sobre la Base de Datos y en el Anexo D.2. Diseño de los módulos VO y DAO. la información detallada sobre los DAO y VO.

Este módulo debe permitir el acceso a ciertos recursos sin necesidad de autenticarse, mientras que el acceso a otros recursos si que estarán restringidos a los usuarios autenticados. Ademas este componente está protegido por HTTPS y devolverá las respuestas en formato JSON. Además este módulo permitirá registrar nuevos usuarios en el sistema y es el encargado de asegurar la persistencia en la base de datos. Este módulo debe controlar los casos de fallo e informar a los usuarios de lo ocurrido en la medida de lo posible.

5 Data Access Object
6 Value Object

2.2.3. Car Services.

En la figura 6 se puede ver el diseño del componente Car Services.

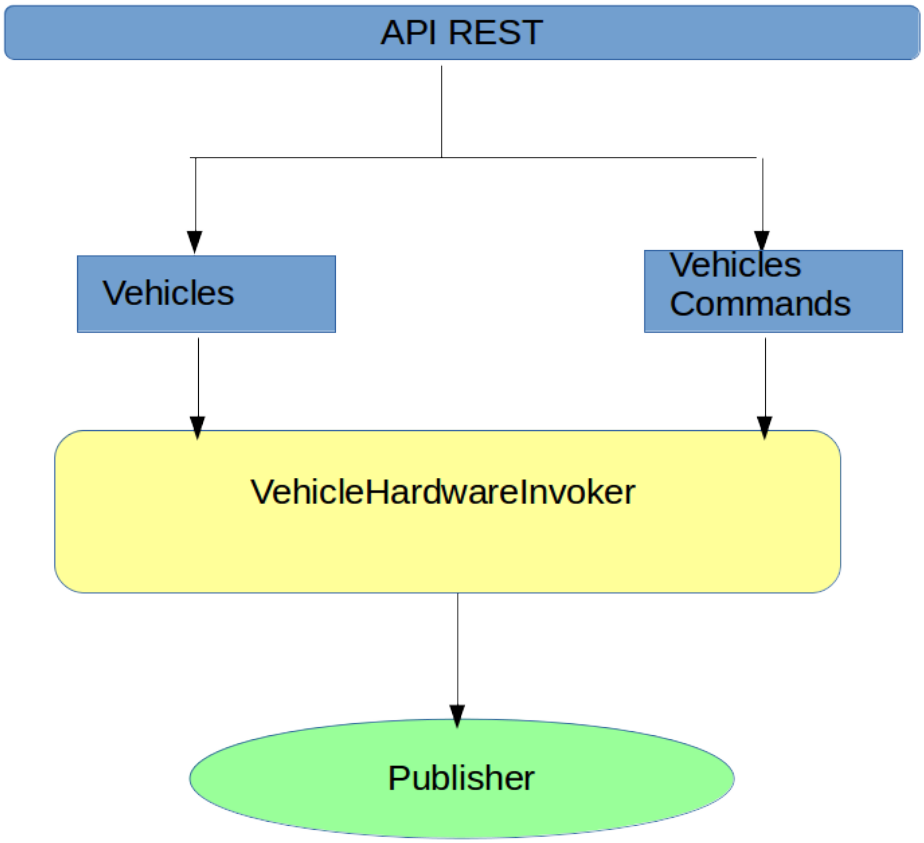


Ilustración 6: Esquema del diseño del módulo Car Services..

Como ya se ha explicado este componente simula la API de Tesla para controlar los vehículos. Se comunica con el componente Renting Services mediante una API REST, en la que también se pueden diferenciar dos grandes módulos.

- **Vehicles**: Recibe las peticiones web relacionadas con la información de los vehiculos.
- **VehiclesCommands**: Recibe las peticiones web que modifican el estado del vehículo.

Ambos módulos se conectan con el módulo *VehicleHardwareInvoker*, este módulo recibe los parámetros de las peticiones web y se comunica con el módulo *Publisher*. Este módulo es el encargado de publicar en el servidor *Mosquitto* los mensajes correspondientes. Los temas que este módulo tiene disponible son:

- **door_lock/{vehicleId}**: tema para cerrar las puertas del vehículo con identificador {vehicleId}. Se publicara un mensaje “door_lock”.
- **door_unlock/{vehicleId}:** tema para cerrar las puertas del vehículo con identificador {vehicleId}. Se publicara un mensaje “door_unlock”.

Como sólo se dispone de una *RapsBerry* se ha supuesto que es el vehículo con id =1, por tanto se ha configurado de tal manera que aunque al componente le llegue otro {vehicleId} siempre se publiquen los mensajes con {vehicleId} =1.
Para acceder a cualquier recurso de este módulo se deberá validar al usuario en el servidor de autenticación *Keycloak.* Este módulo debe permitir la comunicación con el vehículo mediante el protocolo *MQTT*. Además todas sus respuestas estarán en formato *JSON.* Este módulo también esta protegido por HTTPS.

2.2.4. OSVehicle (RapsBerry)

Simulando ser el ordenador de abordo de nuestro OSVehicle la *RapsBerry* contiene un pequeño programa Java denominado mediante el cual se subscribe al servidor *Mosquitto* para consumir los mensajes que sean pertinentes. Cuando los consuma mostrará por la salida estándar el mensaje consumido para confirmar su correcto funcionamiento. La *RapsBerry* simulando ser el OSVheicle esta subscrita a dos temas:

- **door_lock/{vehicleId}:** El tema correspondiente para bloquear la puerta.
- **door_unlock/{vehicleId}:** El tema correspondiente para abrir la puerta.

{vehicleId} es el identificador del vehículo, es decir, cada vehículo estaría subscrito a un par de temas propios.
El programa implementado en la *RapsBerry* se denomina OSVehicle-client, a este programa hay que pasarle dos parámetros al ejecutarlo, el primero es la dirección IP de nuestro broker *Mosquitto* y el segundo es el {vehicleId} del vehículo al que representa, en nuestro caso al solo disponer de una *RapsBerry* el {vehicleId} siempre será 1.
Este programa debe mostrar por la salida estándar el mensaje que consume del broker *Mosquitto* para confirmar que recibe correctamente el mensaje. Como se explica en la sección 4.2. Trabajo futuro. el siguiente paso sería implementar la parte electrónica del vehículo, es decir, una vez consume el mensaje correctamente que se abrieran las puertas de verdad.

2.3. Implementación

Como resultado final se ha implementado un sistema con el que poder controlar y obtener información remotamente de un vehículo. Para desarrollar este sistema se ha utilizado el lenguaje de programación *Java* mediante el IDE[7] *Eclipse[11].*

Para la gestión de dependencias entre módulos y distintas versiones de librerías se ha utilizado la herramienta *Maven[12].*

Para el control de versiones se ha utilizado *Git[13],* en un repositorio de *GitHub[14].* Se ha utilizado *CircleCi[15]* como entorno de integración continua.

Para la creación de las API REST se ha utilizado el protocolo HTTPS[16], el ptrocolo OAUTH2[9] y el servidor Keycloak[6], ademas del framework *RESTEasy[17]. RESTEasy* se puede ejecutar en cualquier contenedor de *Servlet[18],* en nuestro caso se ha elegido *Apache Tomcat 8.5[19].* Para deserializar (proceso mediante el cual vamos a convertir

7 Integrated Drive Electronics

una cadena de texto con una representación de JSON[20] de un objeto en un objeto real de Java) y serializar (convertir un objeto Java en una cadena de texto con su representación JSON) se ha utilizado la librería *Jackson[21].*

Para el mapeo entre la base de datos y el modelo de objetos de nuestro sistema se ha utilizado la librería *Hibernate[22].* Esta librería implementa el estándar JPA[23].

Como base de datos se ha elegido *Maria DB[10].*

Como Broker (Servidor) de *MQTT[7]* se ha elegido *Mosquitto[24].*

Para realizar los test pertinentes de nuestro sistema se ha utilizado la librería *Junit[25].*

Finalmente para el desarrollo de la aplicación web se ha elegido el framework *Ionic 2[8].* Este framework se basa en *Apache Cordova[26],* y nos permite crear aplicaciones móviles híbridas basadas en *HTML5, CSS y JavaScript.* Esta construido con *Sass[27]* y optimizado con *AngularJS[28].*

Para la automatización del sistema se ha utilizado Virtualbox[29], Vagrant[30] y Ansible[31]. Circle github OAUTH KEYclak

2.4. Validación.

Para asegurar el correcto funcionamiento de todos los componentes se han realizado una serie de pruebas tanto por separados como una vez integrados.

Para comprobar el funcionamiento de nuestros servicios web y del servidor de autenticación *Keycloak* se ha utilizado la herramienta *Postman[32]* para comprobar el correcto funcionamiento de las peticiones y respuestas con y sin autenticación. A continuación se realizaron pruebas para verificar el correcto funcionamiento del protocolo *MQTT* y de la persistencia de la base de datos.

Por último se ha procedido a realizar una serie de pruebas sobre la aplicación móvil con el objetivo de comprobar el correcto funcionamiento del sistema completo. A continuación se muestran las pruebas más destacadas, el resto se encuentran en el Anexo E.

Prueba	**Iniciar aplicación sin registrarse**
Contexto	Se quiere comprobar que la API REST permite el acceso a ciertos recursos sin autenticación
Resultado	Se comprueba que efectivamente la API REST permite acceder a algunos recursos sin necesidad de tener Acces-Token

Prueba	**Reservar vehículo sin registrarse**
Contexto	Se quiere comprobar que la API REST no permite el acceso a ciertos recursos sin autenticación
Resultado	Se comprueba que efectivamente para otras recursos la API REST si que requiere un Acces-Token.

Prueba	**Alquilar coche**
Contexto	Se quiere comprobar el correcto funcionamiento del sistema completo.
Resultado	Se comprueba que efectivamente el sistema completo funciona correctamente. Se comprueba que osvehicle-client consume el mensaje de abrir puertas del broker.

Al realizar estas pruebas por separado se han podido ir corrigiendo los errores poco a poco y realizar iteraciones sobre la lista de requisitos hasta llegar a una solución final robusta que cumpla todos los objetivos establecidos.

Además este sistema podría considerarse la primera fase de un proyecto más complejo explicado en la sección 4.2. Trabajo futuro. de esta memoria.

Capítulo 3.

Valoración del trabajo realizado.

Una vez expuesto el trabajo realizado, se procederá a realizar una valoración tanto de los resultados obtenidos como económica.

3.1. Valoración de los resultados obtenidos.

Tras realizar las pruebas pertinentes en nuestro sistema, se ha comprobado que cumple con los requisitos establecidos. Se han cumplido todos los objetivos establecidos:

- Correcto funcionamiento de nuestras API REST.
- Correcto funcionamiento del protocolo *MQTT*.
- Correcto funcionamiento de nuestro servidor de autenticación *Keycloak.*
- Se garantiza la persistencia de la base de datos.
- La aplicación móvil funciona correctamente.

Con estos resultados obtenidos, tal y como se explica en la sección 4.2. Trabajo futuro., se podría empezar una nueva fase de proyecto que consiste en programar la parte electrónica de nuestro OSVehicle para implementar las funcionalidades de nuestra API.

3.2. Valoración económica.

En este apartado se va a realizar una estimación de los costes del trabajo realizado. Para ello se identificaran las fases del proyecto, la gestión de esfuerzos y por último se estimará un presupuesto.

3.2.1. Fases del proyecto.

Durante la realización del proyecto, este se ha dividido en las siguientes fases. A continuación se definen todas las fases del proyecto.

- **Estudio previo**: La primera fase del proyecto. Esta fase consistió en estudiar tanto el contexto en el que se iba a desarrollar el proyecto, como en la lectura y estudio de la documentación necesaria para la comprensión de las posibles tecnologías y herramientas que se usarían en este proyecto.
- **Análisis**: Esta fase consistió en analizar toda la información obtenida en la fase anterior para determinar el entorno, los requisitos y la estructura del sistema.
- **Diseño**: Esta fase consistió en cómo se iba a desarrollar la aplicación, es decir, determinar todos los componentes de la arquitectura y su funcionalidad.
- **Implementación**: En esta fase se desarrolló todo el diseño planteado en la fase anterior.
- **Validación**:Esta fase comenzó con la primera implementación de la API REST y continuó hasta validar todos los requisitos establecidos.

- **Documentación**: Esta fase se ha llevado a cabo durante prácticamente todo el proyecto, aunque cuando más tiempo se ha invertido ha sido en la fase final. Esta fase consiste en documentar todo el sistema que se ha desarrollado.

3.2.2. Gestión de esfuerzos.

A continuación en la figura 7 se muestra el diagrama de Gantt donde se puede ver la distribución del tiempo dedicado a las diferentes tareas identificadas en el apartado anterior.

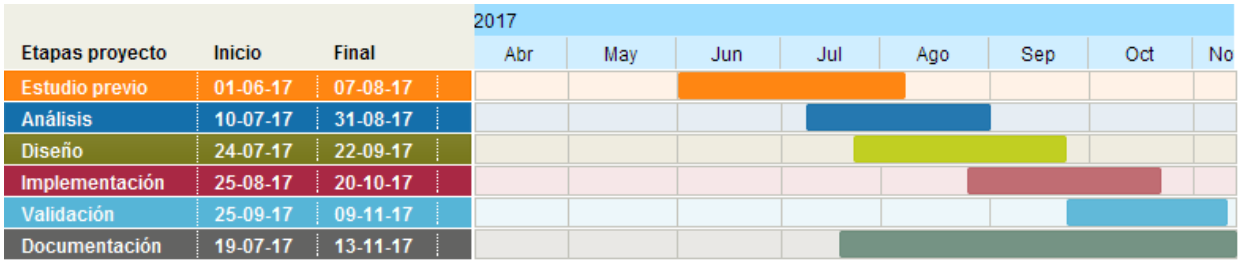


Ilustración 7: Diagrama de Gantt del proyecto.

Como se puede ver en el diagrama de Gantt el estudio previo empieza el día 1 de Junio, día que comenzaron las prácticas en Neodoo Microsystems. También se puede ver como las diferentes fases del del proyecto son lineales, a excepción de la documentación que abarca prácticamente todo el periodo de realización del proyecto. Aunque las fases aparecen constantes y sin parones, en realidad se han hecho dos paradas. Una en verano, en el mes de agosto, y otra en octubre en la semana del Pilar.

A continuación, en la figura 8 se puede ver un gráfico que muestra como se ha repartido el esfuerzo sobre las diferentes fases.

Se puede ver que la fase en la que mas tiempo se ha empleado es en el diseño del sistema, mientras que en la que menos tiempo se ha empleado ha sido en la fase de validación.

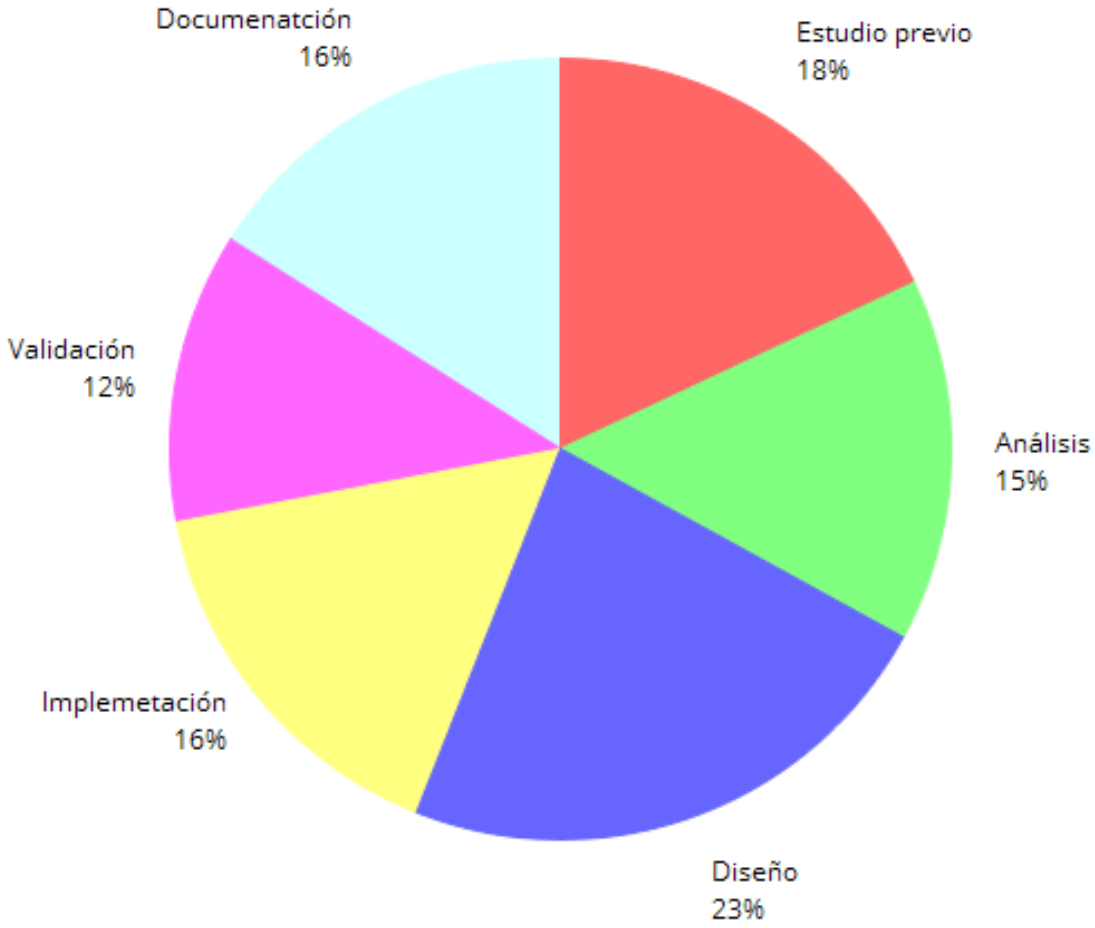


Ilustración 8: Gráfico con el reparto de esfuerzos..

El tiempo invertido en este TFG ha sido de 700 horas. A continuación se muestra una tabla con las diferentes fases del proyecto y el número de horas dedicada a cada una.

Fase	Horas
Estudio previo	126 horas
Análisis	105 horas
Diseño	161 horas
Implementación	112 horas
Validación	84 horas
Documentación	112 horas

3.2.4. Presupuesto.

Por ultimo se va a proceder a estimar un presupuesto para nuestro sistema.

Concepto	Coste
Rapsberry pi 3 Modelo B	37 €
Carcasa para Rapsberry Pi 3 Modelo B	10 €
Compra de dominio .com en Godaddy.com	1€ (oferta por un año)
Amortización de equipo informático	50 €
Horas de estudio previo (126h x 16€/h)	756 €
Horas de Análisis (105h x 19€/h)	840 €
Horas de Diseño (161h x 21€/h)	1610 €
Horas de implemetación (112h x 24€/h)	1344 €
Horas de validación (84h x 21/h)	840 €
TOTAL	**11.942 €**

Como se puede ver en la tabla no todas las fases del proyecto tienen el mismo precio, siendo la más barata el estudio previo y la más cara la implementación. Además la fase de Documentación tampoco se incluye. El precio por hora de las fases cubren otros gastos indirectos como: Calefacción, luz, agua, línea telefónica, alquiler, y otros servicios externos (nómina, jurídico, fiscal, etc).

Por tanto la estimación económica final del sistema sería de 11.942€

Capitulo 4

Conclusiones y trabajo futuro.

Para concluir, en este capítulo se incluyen las conclusiones sobre el proyecto realizado. Se van a plantear varias de las posibles vías de trabajo futuro que se podrían seguir a partir de este proyecto. Para finalizar, se muestra una valoración personal del trabajo realizado.

4.1. Conclusiones.

Tras la finalización del proyecto se puede confirmar que se ha conseguido realizar la capa de acceso al vehículo objetivo. También se puede confirmar que el sistema funciona correctamente ya que ha validado las pruebas pertienentes.

En cuanto a los objetivos conseguidos por parte personal del autor destacan:

- Desarrollar una aplicación en su totalidad. Se ha tenido la responsabilidad de llevar a cabo todos los procesos que intervienen en el desarrollo software de una aplicación, desde el planteamiento inicial hasta la solución final.
- Conocer y aumentar los conocimientos de tecnologías imprescindibles en la actualidad como por ejemplo: *GitHub, Maven o REST.*
- Ser capaz de solucionar los problemas surgidos en el desarrollo de software. Así como crear todo el entorno y sistema.

El sistema desarrollado supone un acercamiento a la posibilidad de poder implementar una capa software para un OSVehicle, ya que se ha comprobado que nuestro sistema funciona correctamente.

4.2. Trabajo futuro.

Tras la finalización del proyecto, se distinguen múltiples vías por las que se puede proseguir el trabajo realizado.

- **Implementación de las funcionalidades en el OSVehicle**: Esta seria la opción de trabajo futuro más importante. Sería necesario adquirir un OSVehicle para poder implementar la capa de electrónica que nos permita realizar las funcionalidades de nuestra API. Esta vía de trabajo futuro es esencial si se quiere conseguir el objetivo de implementar una capa de software en los OSVehicles.

A continuación se muestran las posibles vías de trabajo futuro si se decidiera desarrollar la aplicación móvil de *Carsharing:*

- **Desarrollo de una herramienta administrativa**: Actualmente para modificar la base de datos hay que realizar consultas directamente sobre ella, no existe ninguna herramienta que nos permita administrar nuestra flota de vehículos o

nuestros usuarios, por tanto es necesario crear una herramienta para gestionar nuestra aplicación.

- **Validación de los datos del usuario**: actualmente a la hora del registro de los usuarios solo se imponen algunas restricciones básicas en los datos que introduce, por ejemplo, el número de tarjeta tiene que ser de la longitud adecuada y solo pueden ser números, pero no se comprueba que la tarjeta exista y sea válida. Por tanto es necesario validar todos los datos del usuario.
- **Implementación de una pasarela de pago**: Si la aplicación saliera al mercado seria necesario incluir una pasarela de pago, para, poder cobrar al cliente el importe que tenga que pagar en cada viaje.
- **Creación de una herramienta de tracking**: actualmente en la nuestra aplicación móvil solo se puede saber la calle origen y la calle destino de un determinado viaje. Si se implementara esta herramienta de tracking se podría mostrar al usuario el recorrido completo que ha hecho en cada viaje.

4.3. Valoración personal.

La realización de este TFG en la empresa Neodoo Microsystems S.L. me ha supuesto una gran experiencia, tanto en el ámbito personal como en el profesional.

Cuando me llegó la hora de elegir Trabajo Fin de Grado no tenía nada claro si realizarlo por mi cuenta o realizarlo en unas prácticas. Tampoco sabía sobre que podía hacerlo, ya que tampoco tenia claro que rama de la carrera era la que me gustaba más. Finalmente me decidí por realizar unas prácticas para así poder introducirme en el mundo laboral y hacerme una idea de lo que me espera después de la universidad. Por tanto me reuní con Javier Zarazaga para explicarle mi caso y que me diera su opinión. Javier me puso en contacto con varias empresas. De estas empresas la que mas me llamo la atención fue Neodoo Microsystems, ya que tras una reunión con Francisco Javier Solans me explicó el proyecto OSVehicle y me comento la posibilidad de poder implementar la capa de software para estos. Este proyecto me llamo mucho la atención y me pareció el más interesante. Aunque para ello tendría que trabajar con muchas tecnologías que desconocía. Por tanto tuve que emplear muchas horas en el estudio previo, estudiando nuevas tecnologías que algunas de ellas finalmente no se han utilizado. Por todo esto esta experiencia me ha resultado muy positiva ya que me ha permitido introducirme en el mundo laboral y me ha permitido obtener la experiencia necesaria para poder afrontar nuevos retos. Además me ha servido para comprobar que los conocimientos adquiridos durante la carrera no se restringen únicamente a lo aprendido en las asignaturas, si no que me ha permitido desarrollar una serie de habilidades que podrían considerarse propias de un ingeniero, como son: perseverancia, aprendizaje continuo y capacidad para solucionar los problemas.

Anexo A.

OSVehicle.

OSVehicle es una empresa con sede en Italia que trata de minimizar el coste que supone para las nuevas empresas adentrarse en la industria del automóvil, por los altos costes en capital que supone. La revolución del vehículo eléctrico está creando nuevas oportunidades y está afectando a la industria con la incorporación de nuevas empresas que están acelerando el ritmo de la innovación.

Así nace el proyecto *TABBY,* un proyecto de vehículo construido utilizando elementos hardware y software Open Source. Diseñado para ser ensamblado e manera sencilla y por cualquier persona, se necesita menos de 1 hora para montarlo completamente y salir montado en él. *Urban TABBY* es la evolución del mismo, y cuenta con las partes necesarias para circular legalmente en las calles (por el momento sólo en Europa).

Puede ser armado con 2 o 4 asientos, lo que supone que es muy modificable y personalizable. Además al ser un proyecto open Source todos pueden colaborar, pueden descargar los planos del vehículo en 3D modificarlos y compartirlos con la comunidad.

En la figura 9 podemos ver como de un mismo chasis se pueden crear diseños muy diferentes.

Ilustración 9: Esquema de diferentes diseños de OSVehicle con el mismo chasis..

A.1. Estadísticas de TABBY.

Algunas estadisticas de este vehículo son:

- La versión 100% eléctrica tiene una autonomía de entre 90Km y 100 Km entre carga y carga.
- Su velocidad máxima alcanza los 80 Km/H.
- La versión de motor híbrido tiene un alcance de 400 Km.
- Se puede montar en menos de una hora.
- Viene embalado en cajas fácilmente transportables y no necesita herramientas especiales para el montaje.
- El precio de los modelos completamente terminados cuestan entre 4.000€ y 6.000€.

A.2. Grupo Renault y OSVehicle

Con motivo de la celebración del CES (Consumer Electronics Show) 2017, el Grupo Renault y sus socios, OSVehicle, ARM, Pilot y Sensoria presentaron sus proyectos de “open innovation” y “open source”. Fruto del encuentro entre tecnología y transporte, estos proyectos ilustran nuevas formas de imaginar la movilidad. Renault y sus socios sacan partido a la actual transformación de la industria automovilística mediante las modificaciones de hardware o software, nuevos elementos personalizables u oportunidades que ofrece el mercado de recambio.

Apoyándose en el trabajo realizado para *Twizy*[8], nace *POM* (Platform Open Mind), el primer vehículo Open Source producido a gran escala en el mundo. *POM* es un vehículo compacto y ligero desprovisto de piezas de carrocería y dotado de una plataforma automovilística Open Source. Destinado a las start-ups, laboratorios independientes, clientes privados e investigadores, permite a terceros copiar y modificar el software existente para crear un vehículo eléctrico totalmente personalizable.

Gracias a la colaboración entre Renault y OSVehicle sobre el desarrollo de una plataforma Open Source de fácil acceso para la comunidad, OSVehicle propone asimismo servicios de diseño e ingeniería “*a la carta”* para una personalización completa. OSVehicle, que reúne un amplio ecosistema de emprendedores, desarrolladores, diseñadores e ingenieros, simplifica la fabricación, el intercambio, la distribución y la modificación hardware de los vehículos eléctricos.

8 Vehículo biplaza eléctrico producido por el fabricante Reanult.

A.3. Impacto medioambiental

*TABBY, a*l igual que Tesla Model S, X, 3 y Nissan Leaf, son 100% eléctricos. Esto significa que se habilita a nuevas marcas para que hagan su propia marca EV[9] más fácil e introduzcan vehículos 100% eléctricos en el mercado, fáciles de adaptar a los estándares disponibles en los sistemas de carga locales.

Fabricar vehículos 100% eléctricos no es el único paso que dar para salvar el planeta. Es solo el primer paso. El mayor problema es lo que sucede una vez que el vehículo termina su ciclo de vida. El problema es que la mayoría de los vehículos están diseñados para el basurero. En la figura 10 se puede ver reflejado este problema.

Pero, afortunadamente, se ha resulto este problema. Los OSVehicles son modulares, lo que significa que son reemplazables y tienen capacidad de actualización, lo que significa una vida ultralarga.

Ilustración 10: Imagen de un cementerio de coches.

9 Vehículo Eléctrico.

Anexo B.

API de Tesla.

Se ha utilizado la documentación no oficial de Tim Dorr[5].

De acuerdo con esta documentación la lista de todas las funcionales para monitorizar y controlar el modelo S de Tesla remotamente son:

- Autenticación
  - Tokens
    - **Get An Access Token**:
      - Descripción: Realiza el inicio de sesión, devuelve un acces_token que se transfiere como un encabezado con todas las solicitudes futuras para autentificar al usuario.
      - POST
        - https://owner-api.teslamotors.com/oauth/token
      - Atributos:
        - Grant_type: String
        - Client_id: String
        - Client_secret: String
        - Email: String
        - Password: String
      - Response:
        - Headers: Content-Type:application/json
        - Body:

```
{
  "access_token": "abc123",
  "token_type": "bearer",
  "expires_in": 7776000,
  "created_at": 1457385291,
  "refresh_token": "cba321"
}
```

- Vehículos
  - Colección de vehículos
    - **List All Vehicles**:
      - Descripción: Lista todos los vehículos disponibles.
      - GET
        - https://owner-api.teslamotors.com/api/1/vehicles
      - Request:
        - Headers: Authorization: Bearer {access-token}
      - Response:
        - Headers: Content-Type:application/json
        - Body:


```
{
  "response": [
    {
      "color": null,
      "display_name": null,
      "id": 321,
      "option_codes":
"MS01,RENA,TM00,DRLH,PF00,BT85,PBCW,RFPO,WT19,IBMB,I
DPB,TR00,SU01,SC01,TP01,AU01,CH00,HP00,PA00,PS00,AD02,
X020,X025,X001,X003,X007,X011,X013",
      "user_id": 123,
      "vehicle_id": 1234567890,
      "vin": "5YJSA1CN5CFP01657",
      "tokens": [
        "x",
        "x"
      ],
      "state": "online"
    }
  ],
  "count": 1
}
```

- Estados y Configuraciones
  - **Mobile Access**:
    - Descripción: Determina si el acceso móvil al vehículo esta activo.
    - GET
      - https://owner-api.teslamotors.com/api/1/vehicles/{vehicle_id}/mobile_enabled
    - Parámetros:
      - vehicle_id: ID del vehículo.
    - Request:
      - Headers: Authorization: Bearer {access-token}
    - Response
      - Headers: Content-Type:application/json
      - Body:

```
{
  "response": true
}
```

  - **Charge State**:
    - Descripción: Devuelve el nivel de batería.
    - GET
      - https://owner-api.teslamotors.com/api/1/vehicles/{vehicle_id}/data_request/charge_state
    - Parámetros:
      - vehicle_id: ID del vehículo.
    - Request:
      - Headers: Authorization: Bearer {access-token}
    - Response
      - Headers: Content-Type:application/json

- Body:


```
{
  "response": {
    "charging_state": "Complete",  // "Charging", ??
    "charge_to_max_range": false,  // current std/max-range setting
    "max_range_charge_counter": 0,
    "fast_charger_present": false, // connected to Supercharger?
    "battery_range": 239.02,       // rated miles
    "est_battery_range": 155.79,   // range estimated from recent
driving
    "ideal_battery_range": 275.09, // ideal miles
    "battery_level": 91,           // integer charge percentage
    "battery_current": -0.6,       // current flowing into battery
    "charge_starting_range": null,
    "charge_starting_soc": null,
    "charger_voltage": 0,          // only has value while charging
    "charger_pilot_current": 40,   // max current allowed by charger &
adapter
    "charger_actual_current": 0,   // current actually being drawn
    "charger_power": 0,            // kW (rounded down) of charger
    "time_to_full_charge": null,   // valid only while charging
    "charge_rate": -1.0,           // float mi/hr charging or -1 if not
charging
    "charge_port_door_open": true
  }
}
```


- **Climate Settings**
  - Descripción: Devuelve la temperatura actual y el estado del controlador climático.
  - GET
    - https://owner-api.teslamotors.com/api/1/vehicles/{vehicle_id}/data_request/climate_state
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}

- Response
  - Headers: Content-Type:application/json
  - Body:


```
{
  "response": {
    "inside_temp": 17.0,          // degC inside car
    "outside_temp": 9.5,          // degC outside car or null
    "driver_temp_setting": 22.6,  // degC of driver temperature setpoint
    "passenger_temp_setting": 22.6, // degC of passenger
temperature setpoint
    "is_auto_conditioning_on": false, // apparently even if on
    "is_front_defroster_on": null, // null or boolean as integer?
    "is_rear_defroster_on": false,
    "fan_status": 0               // fan speed 0-6 or null
  }
}
```


- **Driving and Position**
  - Descripción: Devuelve el estado de conducción y posición del vehículo.
  - GET:
    - https://owner-api.teslamotors.com/api/1/vehicles/{vehicle_id}/data_request/drive_state
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:
    -


```
{
  "response": {
    "shift_state": null,          //
    "speed": null,                //
    "latitude": 33.794839,        // degrees N of equator
    "longitude": -84.401593,      // degrees W of the prime meridian
    "heading": 4,                 // integer compass heading, 0-359
    "gps_as_of": 1359863204       // Unix timestamp of GPS fix
  }
}
```

- **GUI Settings**
  - Descripción: Devuelve información diversa sobre la configuración de la GUI del automóvil.
  - GET:
    - https://owner-api.teslamotors.com/api/1/vehicles/{vehicle_id}/data_request/gui_settings
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "gui_distance_units": "mi/hr",
    "gui_temperature_units": "F",
    "gui_charge_rate_units": "mi/hr",
    "gui_24_hour_time": false,
    "gui_range_display": "Rated"
  }
}
```


- **Vehicle State**
  - Descripción: Devuelve el estado físico del vehículo, como qué puertas están abiertas.
  - GET:
    - https://owner-api.teslamotors.com/api/1/vehicles/{vehicle_id}/data_request/vehicle_state
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:

- Headers: Authorization: Bearer {access-token}

- Response
  - Headers: Content-Type:application/json
  - Body:

```
{
  "response": {
    "df": false,                // driver's side front door open
    "dr": false,                // driver's side rear door open
    "pf": false,                // passenger's side front door open
    "pr": false,                // passenger's side rear door open
    "ft": false,                // front trunk is open
    "rt": false,                // rear trunk is open
    "car_verson": "1.19.42",      // car firmware version
    "locked": true,             // car is locked
    "sun_roof_installed": false,  // panoramic roof is installed
    "sun_roof_state": "unknown",
    "sun_roof_percent_open": 0,   // null if not installed
    "dark_rims": false,           // gray rims installed
    "wheel_type": "Base19",       // wheel type installed
    "has_spoiler": false,         // spoiler is installed
    "roof_color": "Colored",      // "None" for panoramic roof
    "perf_config": "Base"
  }
}
```

- Comandos del Vehículo
  - **Wake Up Car**
    - Descripción: Despierta el vehículo del estado de reposo. Es necesario para obtener algunos datos del vehículo.
    - POST
      - https://owner-api.teslamotors.com/api/1/vehicles/{vehicle_id}/wake_up
    - Parámetros:
      - vehicle_id: ID del vehículo.
    - Request:
      - Headers: Authorization: Bearer {access-token}

- Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Set Valet Mode**
    - Descripción: Activa o desactiva el modo valet con un PIN para desactivarlo desde el interior del automóvil. El modo Valet limita la velocidad y potencia del vehículo. También desactiva la configuración de Homelink, Bluetooth y Wifi y la capacidad de deshabilitar el acceso móvil al vehículo. También oculta tus lugares favoritos tu hogar y tu trabajo en la navegación.
    - POST
        - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/set_valet_mode?on=true&password=1234
    - Parámetros:
        - vehicle_id: ID del vehículo.
        - On: Activar o desactivar el modo Valet.
        - Passwrod: Pin del modo Valet.
    - Request:
        - Headers: Authorization: Bearer {access-token}
    - Response
        - Headers: Content-Type:application/json
        - Body:

```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Reset Valet Pin:**
  - Descripción: Restablece el PIN establecido para el modo valet, si está configurado.
  - Post:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/reset_valet_pin
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Open Charge Port:**
  - Descripción: Abre el puerto de carga.
  - POST:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/charge_port_door_open
  - Parámetros:
    - vehicle_id: ID del vehículo.

- Request:
  - Headers: Authorization: Bearer {access-token}

- Response
  - Headers: Content-Type:application/json
  - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Set Charge Limit to Standard**:
  - Descripción: Establece el modo de carga a estándar. (90% según la última actualización).
  - POST:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/charge_standard
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```

- **Set Charge Limit to Max Range**
  - Descripción: Establece el modo de carga al rango máximo. (100% según la última actualización). Utilizar con moderación.
  - POST:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/charge_standard
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- Set Charge Limit
  - Descripción: Establece el limite de carga al porcentaje deseado.
  - POST:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/set_charge_limit?percent=limit_value
  - Parámetros:
    - vehicle_id: ID del vehículo.
    - limit_value: porcentaje de carga.
  - Request:
    - Headers: Authorization: Bearer {access-token}

- Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Start Charging**
    - Descripción: Comienza a cargar. Debe estar enchufado, tener energía disponible y no haber alcanzado su límite de carga.
    - POST:
        - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/charge_start
    - Parámetros:
        - vehicle_id: ID del vehículo.
    - Request:
        - Headers: Authorization: Bearer {access-token}
    - Response
        - Headers: Content-Type:application/json
        - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```

- **Stop Charging**
  - Descripción: Para de cargar. Debe estar cargando.
  - POST
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/charge_stop
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Flash Ligths**
  - Descripción: Enciende las luces.
  - POST:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/flash_lights
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}

- Response
  - Headers: Content-Type:application/json
  - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Honk Horn**
  - Descripción: Hace sonar el claxon.
  - POST:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/honk_horn
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```

- **Unlock Doors**
  - Descripción: Desbloquea las puertas del vehículo.
  - POST:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/door_unlock
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Lock Doors**
  - Descripción: Bloquea las puertas del vehículo.
  - POST:
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/door_lock
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}

- Response
  - Headers: Content-Type:application/json
  - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Set Temperature**
  - Descripción: Establece la temperatura para el sistema HVAC.
  - POST:
  - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/set_temps?driver_temp=driver_temp&passenger_temp=passenger_temp
  - Parámetros:
    - vehicle_id: ID del vehículo.
    - driver_temp: temperatura del asiento del conductor.
    - passenger_temp: temperatura del asiento del acompañante
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```

- **Start HVAC System**
  - Descripción: Inicia el sistema de control del clima. Se enfriará o calentará automáticamente, dependiendo de la temperatura establecida.
  - POST:
  - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/auto_conditioning_start
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Stop HVAC System**
  - Descripción: Para el sistema de control del clima.
  - POST:
  - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/auto_conditioning_stop
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}

- Response
  - Headers: Content-Type:application/json
  - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Move Pano Roof:**
  - Descripción: Controla el techo solar del vehículo, si dispone de él.
  - POST
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/sun_roof_control?state=state&percent=percent
  - Parámetros:
    - vehicle_id: ID del vehículo.
    - State: Estado de apertura del techo solar. Open, close, vent o comfort
    - percent: Porcentaje de apertura del techo solar.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```

- **Remote Start**
  - Descripción: Arranca el automóvil para conducir sin llave. Debe empezar a conducir dentro de los 2 minutos posteriores a la emisión de esta solicitud.
  - POST
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/remote_start_drive?password=password
  - Parámetros:
    - vehicle_id: ID del vehículo.
    - Password: Password del usuario.
  - Request:
    - Headers: Authorization: Bearer {access-token}
  - Response
    - Headers: Content-Type:application/json
    - Body:


```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```


- **Open Trunk**
  - Descripción: Abre el maletero del vehículo.
  - POST
    - https://private-anon-d79b2d7ec9-timdorr.apiary-mock.com/api/1/vehicles/{vehicle_id}/command/trunk_open
  - Parámetros:
    - vehicle_id: ID del vehículo.
  - Request:
    - Headers: Authorization: Bearer {access-token}

- Response
  - Headers: Content-Type:application/json
  - Body:

```
{
  "response": {
    "result": true,
    "reason": ""
  }
}
```

Anexo C.

Análisis.

C.1. Análisis del protocolo MQTT.

C.1.1.¿Por que se ha elegido MQTT?

MQTT es un protocolo abierto, sencillo, ligero y fácil de implantar.
Es ideal para responder a las siguientes necesidades:

- Está especialmente adaptado para utilizar un ancho de banda mínimo
- Es ideal para utilizar redes inalámbricas
- Consume muy poca energía
- Es muy rápido y posibilita un tiempo de respuesta superior al resto de protocolos web actuales
- Permite una gran fiabilidad si es necesario
- Requiere pocos recursos procesadores y memorias

C.1.2.¿Cómo funciona?

MQTT es un servicio de publicación/suscripción TCP/IP sencillo y sumamente ligero. Se basa en el principio cliente/servidor.

El servidor, llamado *broker*, recopila los datos que los *publishers (*los objetos comunicantes) le transmiten. Determinados datos recopilados por el *broker* se enviarán a determinados *publishers* que previamente así se lo hayan solicitado al *broker.*

El principio de intercambio se parece mucho al de Twitter. Los *publishers* envían los mensajes a un canal llamado *topic*. Los *subscribers* (suscriptores) pueden leer esos mensajes. Los *topics* (o canales de información) pueden estar distribuidos jerárquicamente de forma que se puedan seleccionar exactamente las informaciones que se desean.

Los mensajes enviados por los objetos comunicantes pueden ser de todo tipo pero no pueden superar los 256 Mb.

C.1.3 ¿por que se ha elegido Mosquitto?.

El broker Mosquitto es parte del grupo de trabajo Eclipse IoT (Una colaboración industrial de compañías que invierten y promueven una comunidad de código abierto para IoT).

Se ha elegido este broker debido a su gran popularidad y su fácil instalación en la mayoría de sistemas, incluido la RapsBerry Pi.

C.2. Análisis de OAUTH2.

Como se ha explicado en la sección 2.2.3. Análisis del Protocolo OAUth2. OAuth2 es un protocolo de autorización que permite a terceros acceder a contenidos propiedad de un usuario sin que el servidor tenga que manejar ni conocer las credenciales del usuario.

A continuación se va a analizar este protocolo paso a paso.

C.2.1. Registro de un cliente en el servidor de recursos

Un cliente debe estar dado de alta en el servidor de recursos al que quiere acceder. Un cliente se identifica con un *client_id* y normalmente con un *client_secret* además de las URL de redirección para usar en el intercambio de tokens y accesos. OAuth2 define dos tipos de cliente:

- **Confidencial:** Son aquellos clientes que son capaces de mantener las credenciales de autenticación seguras y de manera confidencial.
- **Públicos**: Son los clientes que no son capaces de mantener la confidencialidad de las credenciales. Son los clientes web puros o las aplicaciones nativas de terminales y dispositivos.

C.2.2. EndPoints en el proveedor

El proveedor puede definir dos puntos finales para permitir todo el flujo de autorización-validación (en algunos casos los dos son necesarios y en otros, como el nuestro, solo el token-endpoint es necesario). Uno de ellos será el encargado de autentificación y validación de credenciales y el otro será el encargado de expedir tokens de acceso.

- **Authorization end-point**: Este punto de acceso debe ser el encargado de validar clientes y propietarios de recursos. Cuando un cliente quiere acceder a un recurso, es en este punto en el que el proveedor valida tanto el cliente como el propietario y genera concesiones para acceder a los recursos protegidos. Normalmente recibe (aparte de las credenciales necesarias) una url de redirección a la que redirigir el flujo una vez el propietario y el cliente han sido validados y el propietario ha concedido los permisos necesarios.) Cuando esto ocurre, a la url de redirección es a la que se le traspasan las concesiones (normalmente en formato de código).
- **Token end-point:** este punto es el encargado de dar tokens de acceso, normalmente a cambio de concesiones previamente expedidas por el Authorization end-point.

En nuestra aplicación solo es necesario el Token end-point.

C.2.3. Obteniendo permisos para acceder al recurso protegido

Cuando un cliente quiere acceder a recursos de un propietario, lo primero que necesita es solicitar permisos al propietario del recurso. Este paso de denomina Authorization Grant**.**

Hay cuatro tipos de Authorization Grant definidos por OAuth2 y éstos determinan la manera de conseguir un token de accedo válido. A continuación se va a explicar el tipo utilizado por nuestra aplicación.

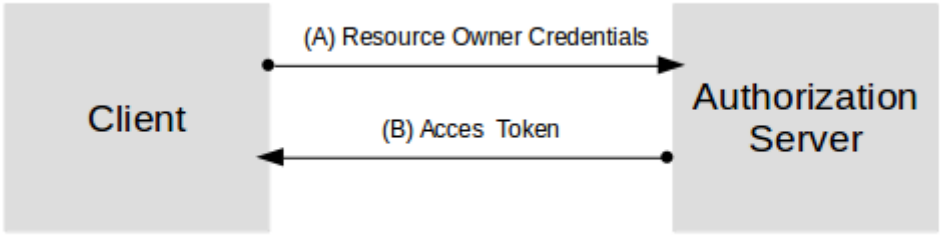


Ilustración 11: Esquema de obtención de credenciales.

Credenciales del cliente: En este escenario únicamente interviene el cliente y el servidor de autorización. Se utiliza cuando es el mismo cliente el que quiere acceder a datos del servidor de autorización sin necesidad de hacerlo en nombre de un propietario de recursos. En la figura 11 se puede ver un esquema.

- (A): El cliente solicita al servidor de autorización un token de acceso enviándole directamente las credenciales del usuario de los recursos.
- (B) El servidor de autorización después de haber validado las credenciales devuelve directamente un token de acceso válido.

En la figura 12 se puede ver un esquema completo de la arquitectura de OAuth2.

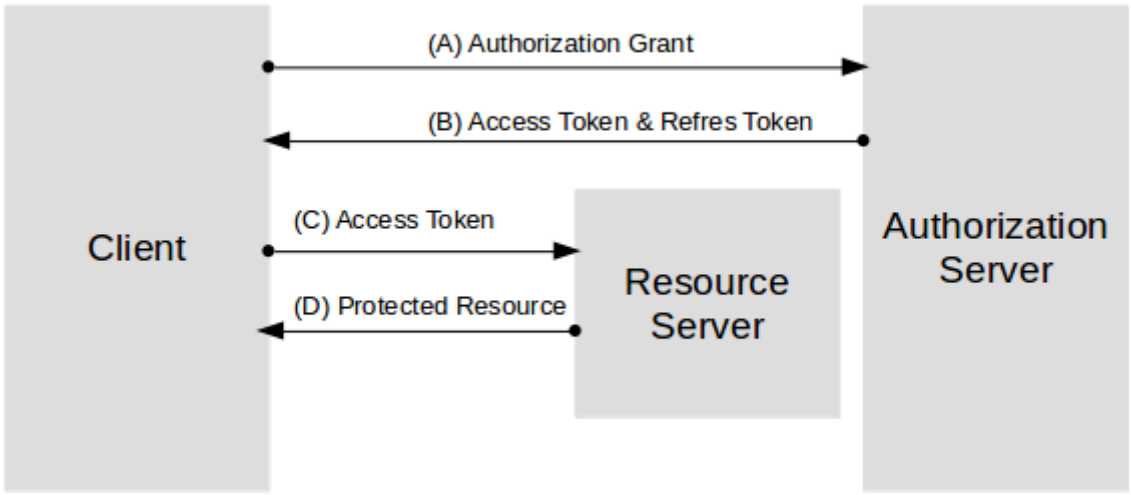


Ilustración 12: Esquema arquitectura OAuth2.

- (A): El cliente pide un token de acceso al proveedor, presentando un Authorization Grant.
- (B): El Authorization Server valida la petición y devuelve un acces token.
- (C): El cliente pide recursos al servidor de recursos presentando el acces token.
- (D): El servidor de recursos valida el acces token y si es válido devuelve los recursos protegidos.

C.3. Análisis de la competencia de la aplicación móvil.

Se ha realizado un análisis de la competencia de la aplicación móvil, es decir, se han estudiado aplicaciones similares ya existentes, aunque por el momento solo se encuentran en las grandes ciudades.

- **CAR2GO**: aplicación de alquiler de coches eléctricos en Madrid.
  - Ventajas:
    - Tiene acuerdos con el ayuntamiento de Madrid para tener plazas de aparcamiento por el centro.
    - Dispone de una gran flota de vehículos.
  - Desventajas:
    - Para registrarse hace falta validar el carnet de conducir en un puesto físico de Madrid.
    - Solo dispone de vehículos de 2 asientos.
- **Emov**: Aplicación de alquiler de coches eléctricos en Madrid.

- Ventajas:
  - Tiene acuerdos con el ayuntamiento de Madrid para tener plazas de aparcamiento por el centro.
  - Dispone de una gran flota de vehículos.
  - Registro completamente online, no hace falta validarlo en un lugar físico.
  - Vehículos de 4 plazas.
- Desventajas:
  - Precios ligeramente mas caros.
  - Menos información y aplicación más básica.
  - Los usuarios reportan gran cantidad de errores de implementación en los comentarios de Google Play.

- **Muving**: Aplicación de alquiler de motos eléctricas en varias ciudades de España.
  - Ventajas:
    - Registro online, no hace falta validarlo en un lugar físico.
    - Precios ligeramente más baratos.
  - Desventajas:
    - Hasta no estar en la moto no puedes saber si dispondrás de 1 casco o 2 (aunque el servicio garantiza 2).
    - Los usuarios reportan gran cantidad de errores de implementación en los comentarios de Google Play.

C.4. Requisitos de la aplicación móvil.

A continuación se muestran los requisitos funcionales y no funcionales de la aplicación móvil.

C.4.1. Requisitos funcionales.

Los requisitos funcionales del sistema describen lo que el sistema debe de hacer.

Código	Descripción
RF-1	La aplicación tiene que poder conectar con el servidor.
RF-2	Los usuarios no registrados podrán acceder a la pantalla principal y podrán ver los coches disponibles, pero no podrán reservarlos ni alquilarlos.
RF-3	Los usuarios podrán registrarse.
RF-4	Los usuarios podrán iniciar sesión mediante su email y contraseña.

RF-5	Los usuarios podrán ver su posición gps en la aplicación.
RF-6	Los usuarios podrán ver los detalles del coche antes de reservarlo o alquilarlo.
RF-7	Los usuarios registrados podrán reservar un coche.
RF-8	Los usuarios registrados podrán liberar su coche reservado.
RF-9	Los usuarios registrados podrán alquilar el coche que tengan reservado o un coche libre.
RF-10	Al alquilar un coche las puertas de este se abrirán.
RF-11	Los usuarios que viajen en un coche podrán ver los detalles del alquiler en curso.
RF-12	Los usuarios que tengan un alquiler en curso podrá finalizar el alquiler.
RF-13	Al finalizar el alquiler se cerraran las puertas del coche.
RF-14	Al finalizar el alquiler se informará al usuario del precio que deberá abonar.
RF-15	Al finalizar el alquiler, la información sobre este sera añadida a la Base de Datos.
RF-16	Los usuarios registrados podrán acceder a su perfil.
RF-17	Los usuarios registrados podrán ampliar la información de un determinado viaje, al seleccionar dicho viaje del listado de viajes de su perfil.

Hay que recordar que los requisitos funcionales 10 y 13 están simulados, es decir, al no disponer de un coche para implementar estas funcionalidades se obtendrá siempre una respuesta simulando que se han abierto y cerrado correctamente las puertas una vez que el mensaje haya sido consumido por la aplicación java en la *RapsBerry.*

C.4.2. Requisitos no funcionales.

Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema.

Código	Descripción
RNF-1	La aplicación debe de funcionar para la versión 5 de android.
RNF-2	La aplicación debe de ser fácil de usar e intuitiva.
RNF-3	El acceso a los recursos debe estar protegido mediante autenticación.
RNF-4	La aplicación debe ser responsive, es decir, se debe adaptar al tamaño de pantalla del dispositivo con el que se accede.
RNF-5	El sistema debe controlar los casos de fallo e informar al usuario en la medida de lo posible
RNF-6	La aplicación funcionará mediante una red de datos o una red Wi-Fi.
RNF-7	La aplicación mostrará la posición gps si está activada la ubicación en el dispositivo

Anexo D.

Diseño.

D.1. Diseño de la Base de Datos

En la figura 13 se puede ver el esquema entidad/relación de la base de datos.

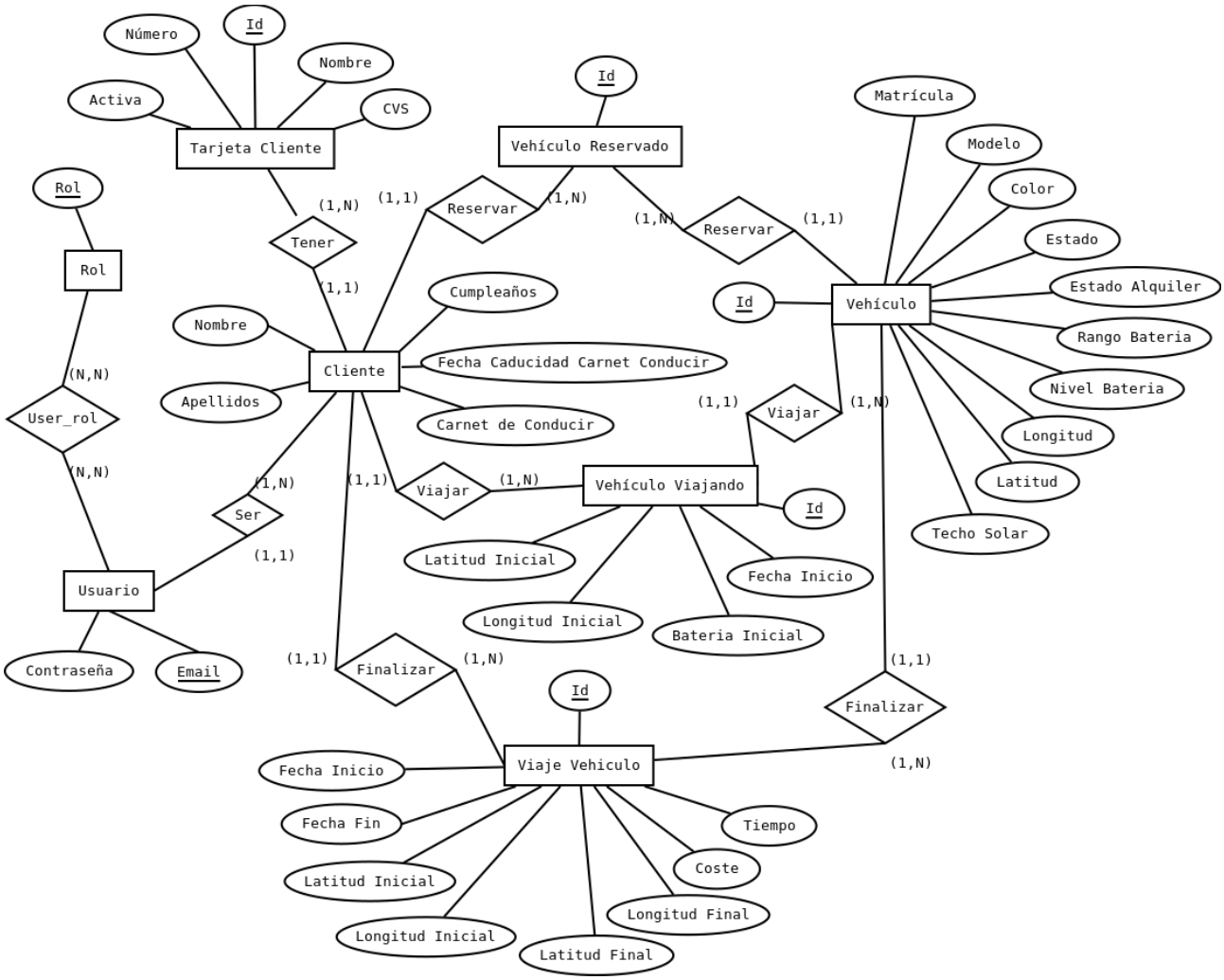


Ilustración 13: Equema Entidad Relación de la Base de Datos.

Como se puede ver existen ocho entidades.

- **Rol**: La entidad Rol se refiere al rol que tendrá el usuario en el sistema. En nuestro proyecto solo se encuentra disponible el rol: “Customer”.
- **Usuario**: Entidad cuyos atributos son: Email y Contraseña.

- **Cliente**: Esta entidad contiene todos los atributos de los clientes. Su clave ajena (Email) proviene de la entidad Usuario.
- **Tarjeta Cliente**: Entidad que contiene la información de la tarjeta de crédito del cliente. Su clave ajena (Email) proviene de la entidad Cliente.
- **Vehículo**: Entidad que contiene todos los atributos de los vehículos.
- **Vehículo Reservado**: Entidad que representa los vehículos que se encuentran reservados por clientes. Su clave primaria es el atributo Id y sus claves ajenas son Email (Cliente) y Vehículo_id (Vehículo). Esta entidad nos permite identificar que usuarios tienen un vehículo reservado.
- **Vehículo Viajando**: Esta entidad representa los vehículos que se encuentran alquilados, es decir, viajando. Su clave primaria es Id y sus claves ajenas son Email (cliente) y Vehículo_id (vehículo), además tiene otros atributos que aportan información sobre el alquiler.
- **Viaje Vehículo**: Entidad que representa los viajes que se han realizado por todos los usuarios. Su clave primaria es Id y sus claves ajenas son Email (cliente) y Vehículo_id (vehículo), además tiene otro atributos que aportan información sobre el viaje.

D.2. Diseño de los módulos VO y DAO.

Estos módulos se encuentran dentro del componente “Renting-services”.

- **Módulo VO**: Este módulo contiene un VO para cada entidad, ya que, cada VO no es más que un objeto que representa una entidad, que tiene como atributos los datos del modelo,con sus correspondientes accesors (getters y setters).
- **Módulo DAO**: Este módulo también contiene un DAO por cada entidad, ya que es el encargado de encapsular el acceso a la base de datos. Por tanto cuando la capa de negocio quiera interactuar con la base de datos tendrá que hacerlo a través de los DAO. Los VO son utilizados por los DAO para transportar los datos desde la base de datos hacia la capa de negocio y viceversa.

D.3. Mapa de navegabilidad de la aplicación móvil.

A continuación en la figura 14 se muestra un esquema del mapa de navegabilidad de la aplicación móvil.

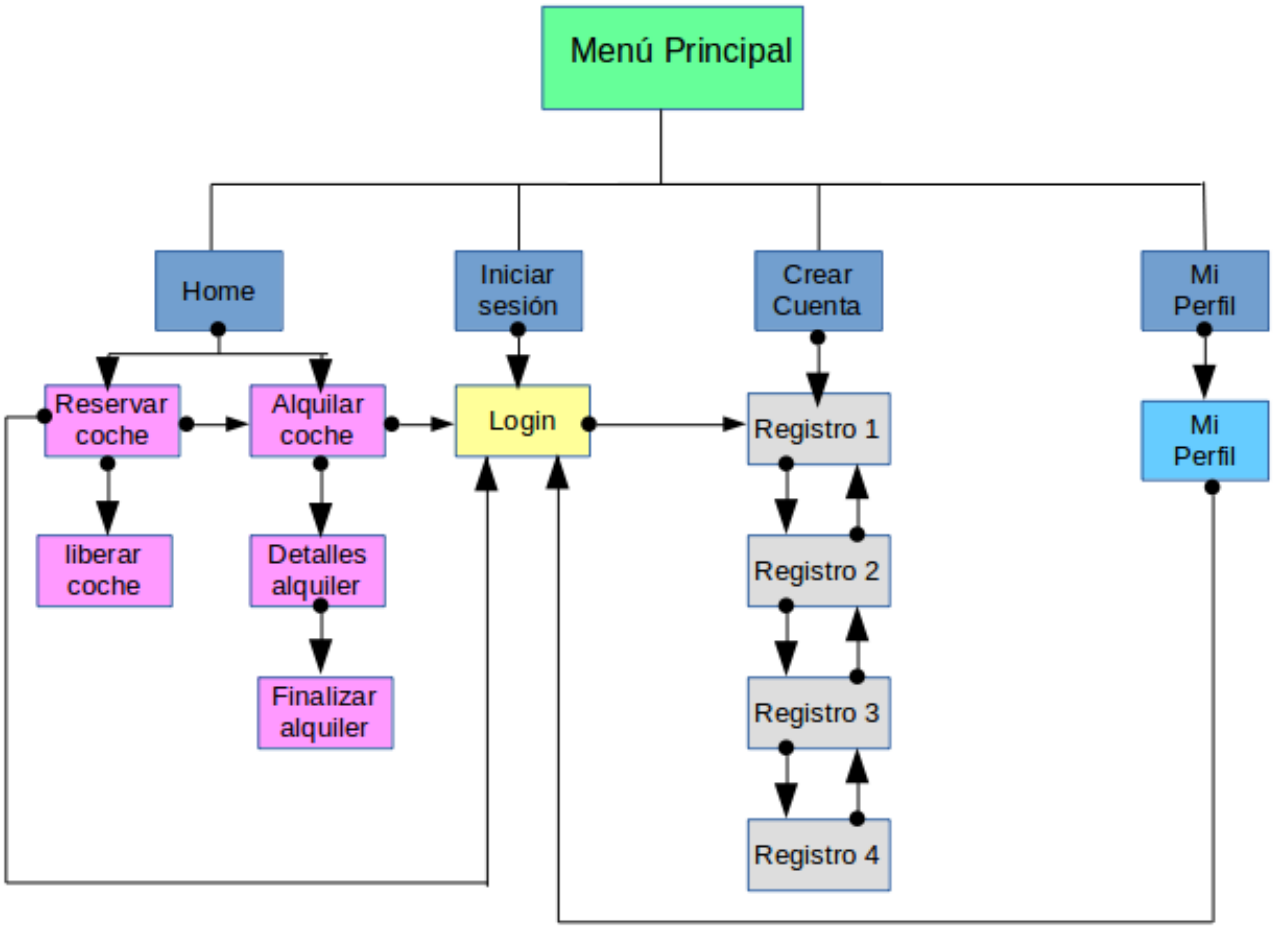


Ilustración 14: Esquema del mapa de navegabilidad.

Como se puede ver en la Figura 14 la aplicación cuenta con un menú que nos permite acceder a las diferentes secciones de nuestra aplicación.

Desde la página principal se puede reservar o alquilar un coche. Para ambas hace falta estar registrado. Una vez reservado se podrá liberar o alquilar. Una vez se ha alquilado un coche se podrán ver los detalles del alquiler y finalizar el alquiler.

Desde la pagina de registro se tendrán que completar los 4 formularios para completar satisfactoriamente este. En todas las paginas se pude navegar hacia adelante o hacia atrás. Para la página de mi Perfil habrá que estar registrado. El menú esta disponible en todas las páginas.

A continuación se muestran los mapas de navegabilidad con más detalle.

D.3.1. Mapa de navegabilidad del menú.

En la figura 15 se puede ver el mapa de navegabilidad del menú.

Ilustración 15: Mapa de navegabilidad del menú.

D.3.2 Mapa de navegación del registro.

En la figura 16 se puede ver el mapa de navegabilidad del registro.

Ilustración 16: Mapa de navegabilidad del Registro.

D.3.3 Mapa de navegación de reservar/alquilar vehículo

En la figura 17 se puede ver el mapa de navegabilidad del reservar/alquilar un vehiculo.

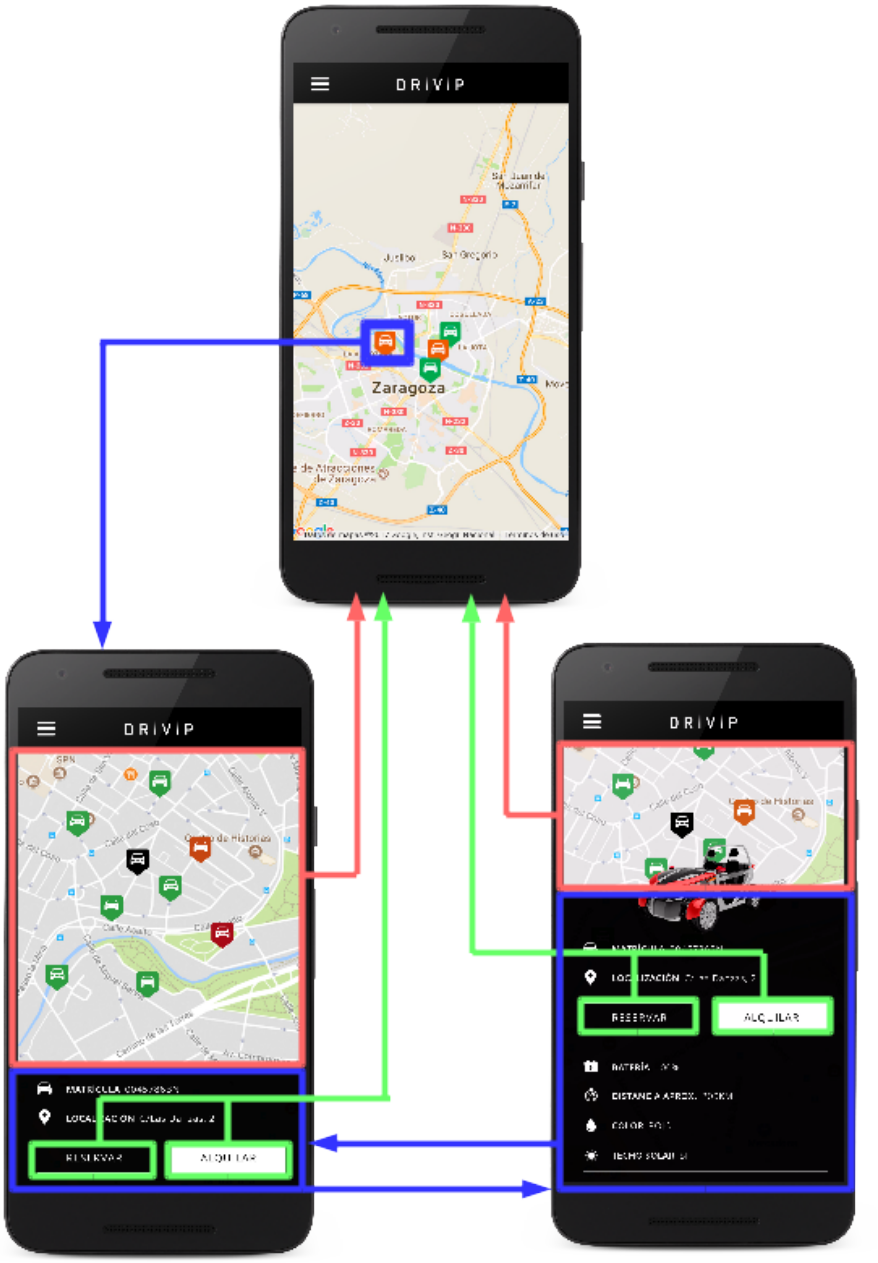


Ilustración 17: Mapa de navegabilidad de reservar/alquilar.

D.4. Diseño de la interfaz de usuario de la aplicación móvil.

A continuación, se detallará el diseño de las páginas de la aplicación móvil.

Se ha intentado mantener el mismo estilo en toda la aplicación. De tal forma que se ha intentado mantener un diseño minimalista en todas las páginas, utilizando iconos que faciliten el feedback con la aplicación. Otro factor que se ha decidido y se ha intentado mantener es diseñar las paginas con el fondo negro y el texto en blanco. La barra de navegación con el menú y el logotipo de la aplicación es común a todas las páginas.

D.2.1 Diseño del icono.

En la figura 18 se puede ver el icono de la aplicación,y en la figura 19 la “Splash Screen” (pantalla de arranque de la aplicación).

Ilustración 18: Icono DRIVIP.

Ilustración 19: Splash Screen.

Como se puede ver en la splash screen aparece el nombre de nuestra aplicación: **DRIVIP**.

Este nombre surge de la combinación de las palabras inglesas: Drive y VIP. Mientras que el diseño del icono, como puede observarse está contenido dentro del nombre de la aplicación y simula una carretera (las “I”) y la marca del gps que señaliza nuestra posición (la “V”).

D.2.2 Diseño de la página principal

En la figura 20 se puede observar la página principal de nuestra aplicación. Como puede verse la página principal consiste en el mapa de Google maps, con unos marcadores en los lugares donde se encuentran los vehículos disponibles. Estos marcadores pueden ser:

- **Verdes**: El coche esta libre y tiene mas del 65% de la batería.
- **Naranjas**: El coche esta libre y tiene menos del 65% de la batería.
- **Negros**: Has reservado o alquilado ese vehículo. Este caso se verá mas adelante.

Por otro lado también en color negro y redondo aparece el marcador de la posición gps del usuario.

Como puede verse nuestra pantalla principal es sencilla, el mapa de nuestra ciudad con nuestra posición gps y los coches que podemos alquilar.

Si seleccionamos un coche se abrirá un “Modal” mostrando la información básica de ese coche, y, permitiendo reservar o alquilar el vehículo al usuario como puede verse en la figura 21.

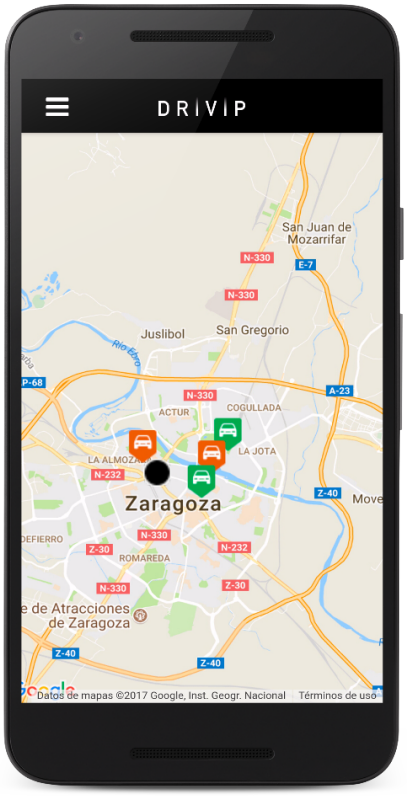


Ilustración 20: Pantalla Principal.

Si volvemos a seleccionar el mismo vehículo, o, seleccionamos el “Modal” que se ha abierto con la información, este se abrirá de nueva mostrando una imagen del vehículo y mostrando mas información sobre este, y, seguirá dando opción a reservarlo o alquilarlo.

Si el vehículo que seleccionamos es el que tenemos reservado (icono negro), el botón de reservar es sustituido por el botón de liberar. En la figura 22 podemos ver los detalles del vehículo.

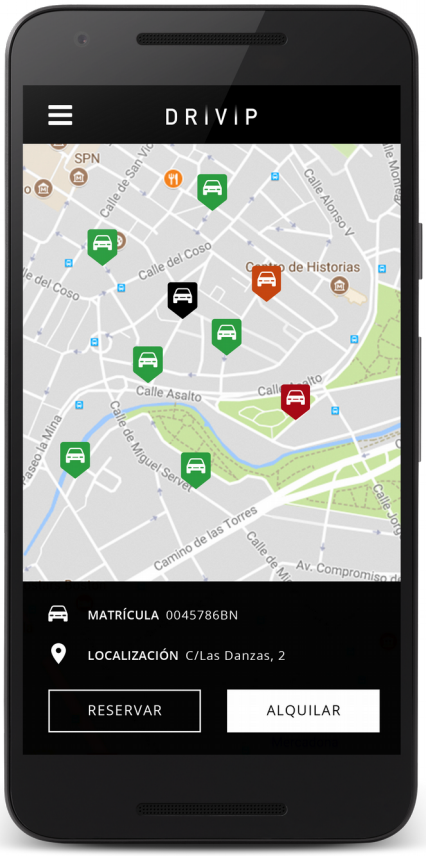


Ilustración 21: Información básica del vehículo.

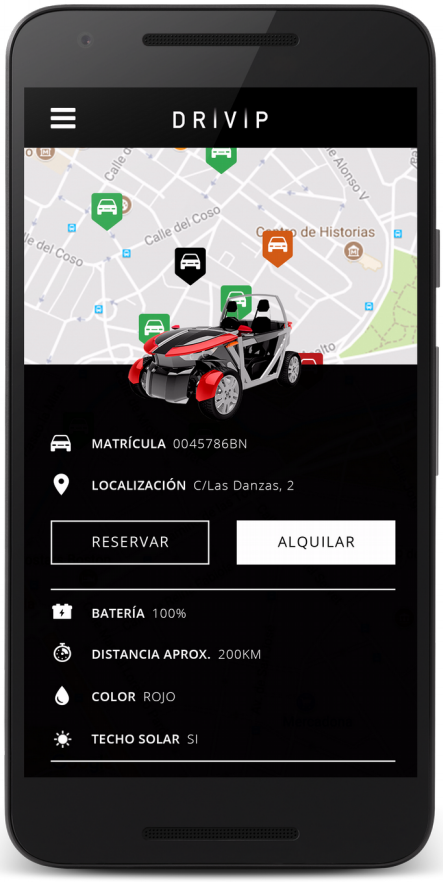


Ilustración 22: Información ampliada del vehículo

D.2.3. Diseño de la página de mi perfil.

La página de Mi perfil tiene el fondo negro y el texto de color blanco. En la parte superior se muestra un saludo al usuario de la aplicación. A continuación se divide en dos partes. La parte superior muestra información general del uso de la aplicación por parte del usuario, es decir:

- Número de viajes.
- Tiempo total.
- Importe total.
- Media de importe de los viajes.

La parte inferior muestra muestra un listado de todos los viajes realizados por el usuario, mostrando la fecha y hora de inicio, y la fecha y hora de finalización. Seleccionando el viaje deseado se abre un desplegable con información más detallada de dicho viaje. Todo esto se puede ver en la figura 23.

Ilustración 23: Pantalla de mi perfil

D.3. Capturas de pantalla finales.

No hay que olvidar que las imágenes anteriores eran prototipos, por tanto el resultado final puede variar ligeramente en algunos casos. A continuación se muestran las capturas de pantalla finales de la aplicación móvil.

La figura 24 es una captura de pantalla de la pantalla principal, mientras que la figura 25 es una captura de pantalla con un vehículo reservado.

Ilustración 24: Captura de la pantalla principal.

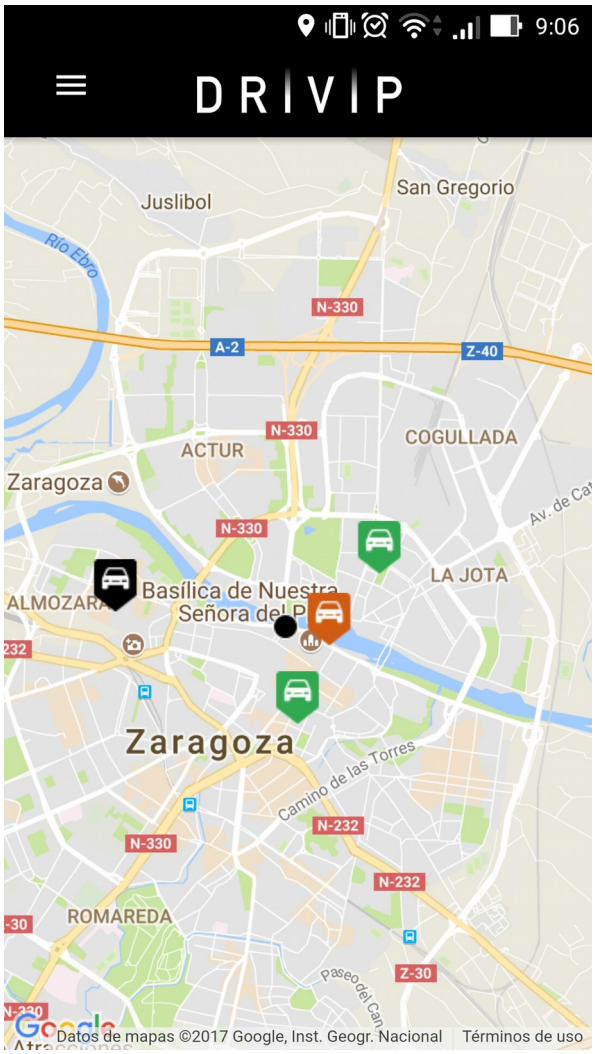


Ilustración 25: Captura de la pantalla principal con un vehículo reservado.

En las figuras 26 y 27 se puede ver las capturas de los detalles de un vehículo al seleccionarlo una vez (figura 26) y al ampliar sus detalles (Figura 27).

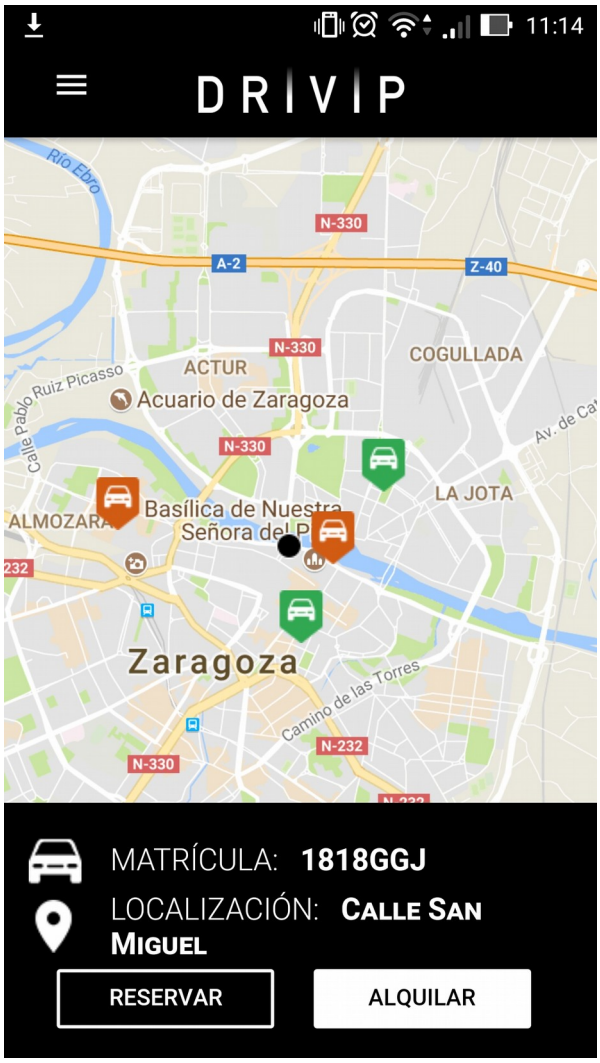


Ilustración 26: Captura de pantalla principal con información básica de un vehículo.

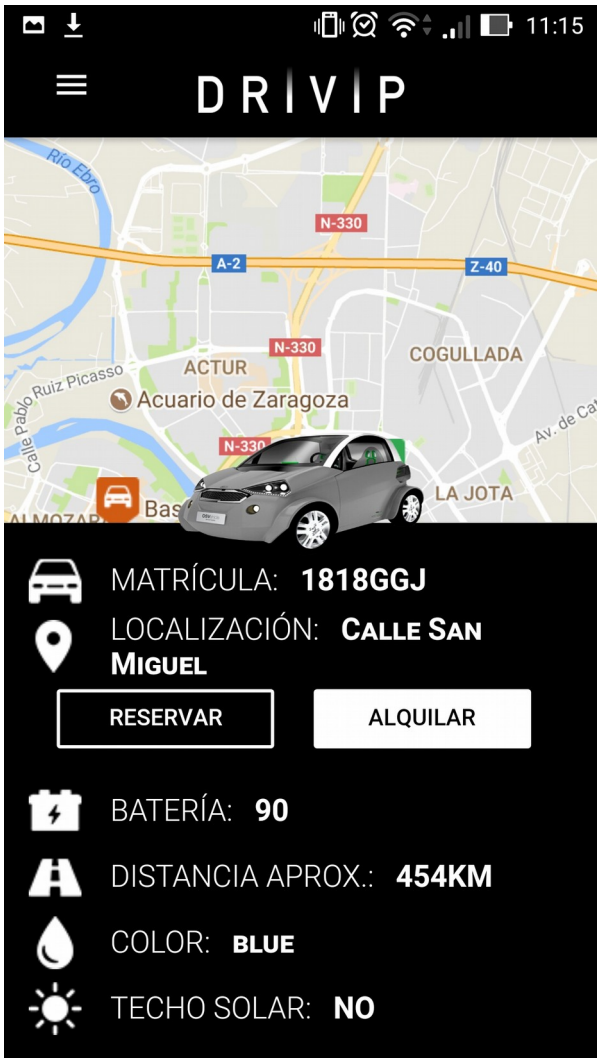


Ilustración 27: Captura de pantalla con información ampliada de un vehículo.

En las Figuras 28 y 29 se puede ver las capturas de pantalla de mi Perfil (Figura 28) y de Mi perfil cuando se amplia la información sobre un trayecto (Figura 29).

Ilustración 28: Captura de pantalla de mi perfil.

Ilustración 29: Captura de pantalla de mi perfil con los detalles de un trayecto.

En las Figuras 30, 31, 32 y 33 se pueden ver las páginas de registro. La página 1 (Figura 30) pide los datos de información personal del usuario, la página 2 (Figura 31) pide información sobre el carnet de conducir del usuario, la página 3 (Figura 32) pide información sobre la tarjeta de crédito, y, por último la página 4 (Figura 33) pide información sobre la contraseña.

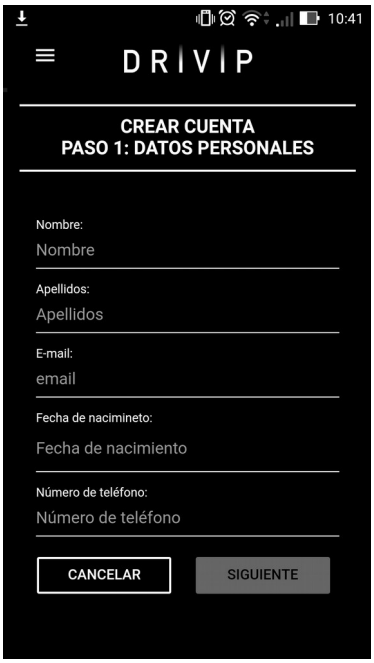


Ilustración 30: Captura de pantalla de la primera pagina del registro.

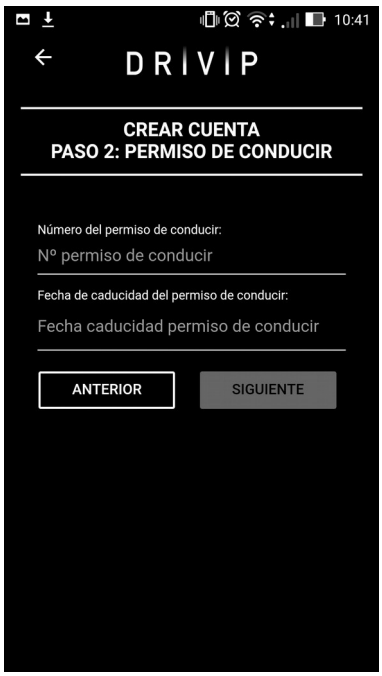


Ilustración 31: Captura de pantalla de la segunda página del registro.

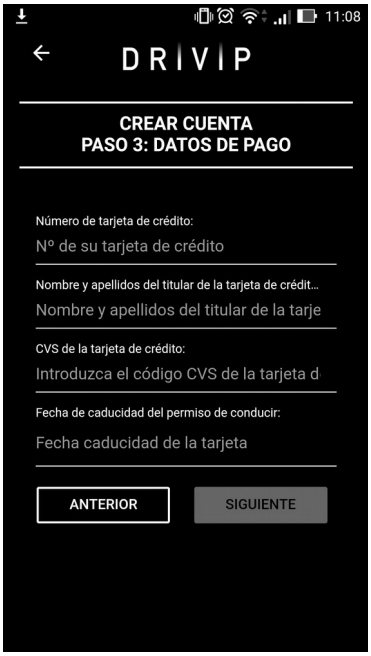


Ilustración 33: Captura de pantalla de la tercera página del registro.

Ilustración 32: Captura de pantalla de la cuarta página del registro.

En las Figuras 34 y 35 se puede ver la página principal al alquilar un vehículo (Figura 34) y la información sobre el alquiler ademas de la posibilidad de finalizar el alquiler (Figura 35) que aparece al seleccionar nuestro vehículo alquilado.

Ilustración 34: Captura de pantalla con un vehículo alquilado.

Ilustración 35: Captura de pantalla con los detalles del alquiler.

Por último en la figura 36 se muestra la pantalla de Login

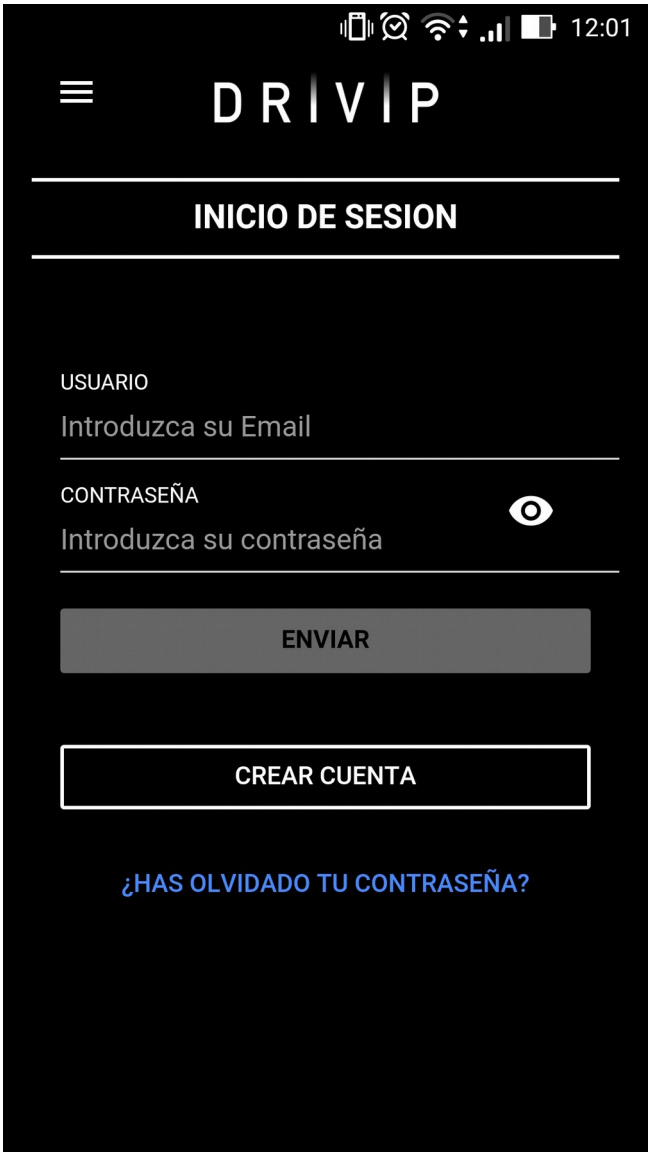


Ilustración 36: Captura de la pantalla de Login.

Anexo E.

Validación de la aplicación móvil.

Para verificar y validar la aplicación móvil se han realizado una serie de pruebas. Estas pruebas se han realizado tanto con el objetivo de comprobar el correcto funcionamiento del sistema de acuerdo a las especificaciones como para comprobar el funcionamiento del sistema ante posibles casos de error. Con estas pruebas se pretende asegurar que la persistencia de datos y la autenticación se comportan correctamente y que se cumplen todos los requisitos funcionales.

A continuación se muestran algunas de las pruebas realizadas.

Prueba	**Iniciar aplicación sin registrarse**
Contexto	Se quiere comprobar que a los usuarios no registrados se les permite ver la lista de vehículos.
Resultado	Se comprueba que efectivamente la API REST permite algunas funcionalidades sin necesidad de tener Acces-Token

Prueba	**Reservar vehículo sin registrarse**
Contexto	Se quiere comprobar que a los usuarios no se les permite reservar o alquilar coches.
Resultado	Se comprueba que efectivamente para otras funcionalidades la API REST si que requiere un Acces-Token.

Prueba	**Registrarse**
Contexto	Se quiere comprobar que un usuario puede registrarse en el sistema.
Resultado	Se comprueba que efectivamente se ha registrado el usuario correctamente ya que se recibe un mensaje indicándolo.

Prueba	**Cumplimiento de las restricciones de registro**
Contexto	Se quiere comprobar que los usuarios tengan que cumplir las condiciones impuestas por los formularios de registro.
Resultado	Se comprueba que efectivamente los usuarios tendrán que rellenar todos los campos y tendrán que cumplir las expresiones regulares que requiera cada campo. (Por ejemplo longitud del número de tarjeta de crédito).

Prueba	**Iniciar sesión**
Contexto	Se quiere comprobar que los usuarios registrados pueden iniciar sesión correctamente
Resultado	Se comprueba que efectivamente los usuarios obtienen el Acces-Token al introducir correctamente sus credenciales.

Prueba	**Cargar el perfil con usuario registrado y no registrado.**
Contexto	Se quiere comprobar que los usuarios registrados pueden cargar su perfil pero los no registrados no.
Resultado	Se comprueba que efectivamente los usuarios registrados pueden cargar su perfil y los no registrados no.

Prueba	**Obtener nuevo Token si el actual ha caducado.**
Contexto	Se quiere comprobar que los usuarios que están registrados pero les ha caducado el Access-Token no necesitan volver a iniciar sesión para realizar acciones que requieren Acces-Token válido.
Resultado	Se comprueba que efectivamente aunque un usuario tengo su Access-Token caducado obtendrá uno nuevo y podrá realizar la petición.

Prueba	**Reservar un coche**
Contexto	Se quiere comprobar que los usuarios registrados pueden reservar un vehículo.
Resultado	Se comprueba que efectivamente los usuarios registrados pueden reservar un vehículo.

Prueba	**Reservar/alquilar un coche teniendo otro coche reservado**
Contexto	Se quiere comprobar que si un usuario tiene un coche reservado no podrá reservar o alquilar otro hasta que no libere el primero.
Resultado	Se comprueba que efectivamente al intentar reservar o alquilar un coche teniendo otro reservado se recibe un mensaje de que se debe liberar primero el coche reservado.

Prueba	**Alquilar coche**
Contexto	Se quiere comprobar que a los usuarios registrados pueden alquilar un coche.
Resultado	Se comprueba que efectivamente los usuarios registrados pueden alquilar un coche (tanto si lo tenían reservado como si no).

Prueba	**Liberar coche**
Contexto	Se quiere comprobar que los usuarios que tienen un coche reservado pueden liberarlo.
Resultado	Se comprueba que efectivamente los usuarios que han reservado un coche pueden liberarlo.

Prueba	**Ver detalles del viaje en curso**
Contexto	Se quiere comprobar que los usuarios que tiene un alquiler en curso pueden ver los detalles de este.
Resultado	Se comprueba que efectivamente se pueden ver los detalles de un alquiler en curso.

Prueba	**Finalizar viaje**
Contexto	Se quiere comprobar que los usuarios con un alquiler en curso pueden finalizarlo cuando quieran.
Resultado	Se comprueba que efectivamente los usuario con un alquiler en curso pueden finalizarlo cuando quieran.

Prueba	**Reservar/alquilar un coche que ya no esta disponible**
Contexto	Se quiere comprobar que no se puede reservar un coche que ya ha sido reservado por otro usuario.
Resultado	Se comprueba que efectivamente si se intenta reservar/alquilar un coche que ya no esta disponible se recibe un mensaje indicando que ya no esta disponible.

Prueba	**Registrarse con un email que ya esta en uso**
Contexto	Se quiere comprobar que un usuario no puede registrarse con un email que ya existe.
Resultado	Se comprueba que efectivamente al intentar registrarse con un email ya existente se obtiene un mensaje de error.

Anexo F.

Manual de configuración.

La configuración del sistema es muy sencilla gracias a la automatización que se ha realizado. Esta manual se ha creado para ejecutarse en un entorno Linux, concretamente se ha utilizado Ubuntu 16.04 para el cual se van a describir los pasos de instalación, pero cualquier otro sería válido. Se parte de que las tecnologías que se han mencionado a lo largo de la memoria están instaladas, en caso contrario se deben seguir las instrucciones que ofrecen en su documentación para instalarlas, se pueden encontrar en las referencias asignadas. Lo primero es alojar el proyecto en el directorio que se desee, para ello si se tiene GitHub instalado simplemente hay que ejecutar el siguiente comando:

git clone https://github.com/neodoo/knightrider.git

Con este comando se descarga el proyecto en el directorio actual. Entramos a este con el comando:

cd knightrider

A continuación se iniciara el entorno virtual mediante el comando:

vagrant up

Este comando levantará una maquina virtual. A continuación habrá que configurar el sistema, para ello ejecutamos:

vagrant provision

A continuación necesitaremos saber la dirección IP de nuestra máquina para ello ejecutaremos el comando:

vagrant ssh

De esta manera nos conectamos a la máquina virtual. Una vez dentro ejecutamos el comando:

ip addr show

Este comando nos mostrara nuestra dirección IP, la cual necesitaremos para cambiarla en la aplicación móvil. Si necesitamos salir de la máquina virtual que hemos creado usaremos el comando :

exit

A continuación ejecutaremos el script change_ip.sh que se encuentra en el directorio principal del proyecto descargado con el siguiente comando:

./change_ip {IP}

Siendo {IP} la ip obtenida. Este scrit cambiará todas las direcciones IP y compilara los servicios web. A continuación tendremos que copiar los ficheros .war en la máquina virtual mediante scp, la password de la máquina es **vagrant**

scp car-services/target/car-services.war vagrant@{IP}:/usr/java/apache-tomcat/webapps
scp renting-services/target/renting-services.war vagrant@{IP}:/usr/java/apache-tomcat/webapps

Con esto ya tenemos toda la configuración. Del sistema. A continuación tendremos que copiar el programa java que muestra por la salida los mensajes recibidos por el servidor *Mosquittto* a la *Rapsberry.* Lo pasaremos también por scp con el comando:

scp OSVehicle-client/target/osvehicle-client {Rapsberry}@{IP}

El siguiente paso es ejecutarlo mediante el comando:

./osvehicle-client {IP} {VehicleId}

Siendo {IP} la dirección Ip del servidor *Mosquitto*y {vehicleId} el id del vehículo que en nuestro caso siempre será 1.

Para terminar, sólo nos queda compilar la aplicación móvil. Para ello volveremos al directorio *renting-mobile-app/drivip* y ejecutaremos el siguiente comando:

ionic cordova build android

Este comando nos creará el fichero .apk que necesitamos para instalar la aplicación. Este fichero se encuentra en el siguiente directorio:

knightrider/renting-mobile-app/drivip/platforms/android/build/outputs/apk

En este directorio se encuentra el fichero android-debug.apk que es el que nos tendremos que mandar a nuestro teléfono móvil. Una vez que se haya pasado al teléfono móvil solo hay que instalarla y ya está lista para usarse.

Anexo G.

Manual de usuario.

En este anexo se explica las características básicas que debe saber un usuario para la utilización de la aplicación móvil.

Al abrir la aplicación se verá la página principal, donde se muestran los vehículos disponibles y nuestra posición GPS. Los vehículos en color verde son aquellos que tienen mas del 65% de batería y los vehículos en color naranja son aquellos que tienen menos. En esta pantalla podemos seleccionar cualquier vehículo y se abrirá un modal con la información básica sobre este, y dos botones, uno para reservarlo y otro para alquilarlo. Si seleccionamos este modal se ampliará y aparecerá mas información y una imagen del modelo del vehículo. Se puede ver en las figuras 37, 38 y 39.

Ilustración 37: Pantalla principal

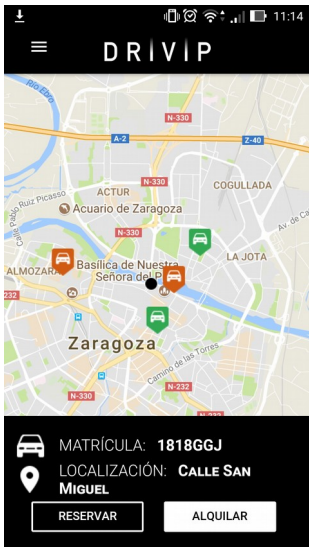


Ilustración 38: Detalles coche seleccionado

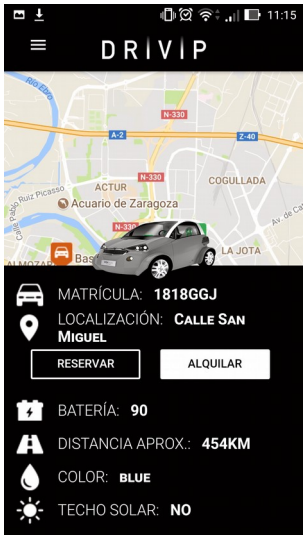


Ilustración 39: Detalles ampliados del coche

Como no estamos registrados, no se nos permitirá ni reservar ni alquilar vehículos. Para ello debemos acceder al menú y a la opción registrarse. Se mostrara un formulario indicando que es el paso uno de cuatro para completar el registro. En este formulario se requiere la información personal del usuario. Hay que completar todos los campos correctamente para que nos permita pasar al siguiente formulario. El segundo formulario requiere la información sobre el carnet de conducir del usuario. Una vez completados correctamente los campos se pasará al tercer formulario. Este formulario requiere la información de pago del usuario. Al completarlo se accede a la información de la contraseña, donde el usuario tiene que introducir su contraseña y confirmarla. Una vez terminado se mostrará un mensaje de éxito o de error dependiendo el caso. Se puede ver en las figuras 40, 41, 42 y 43

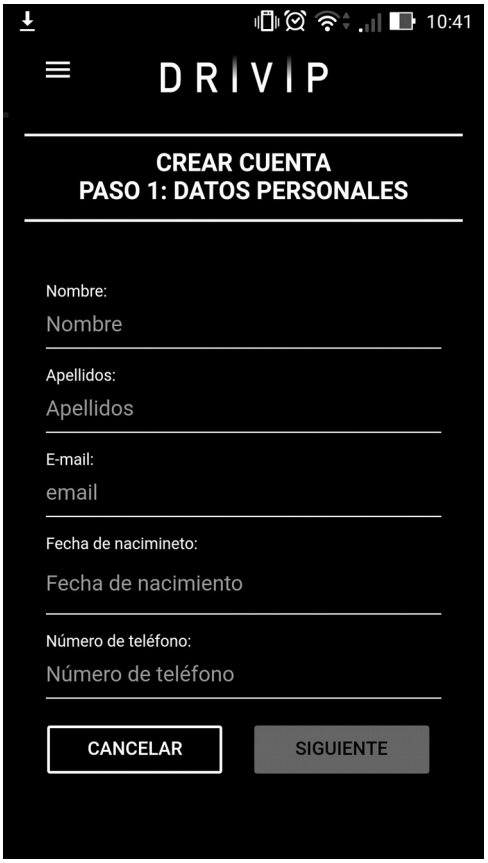


Ilustración 40: Paso 1/4 registro

Ilustración 41: Paso 2/4 registro

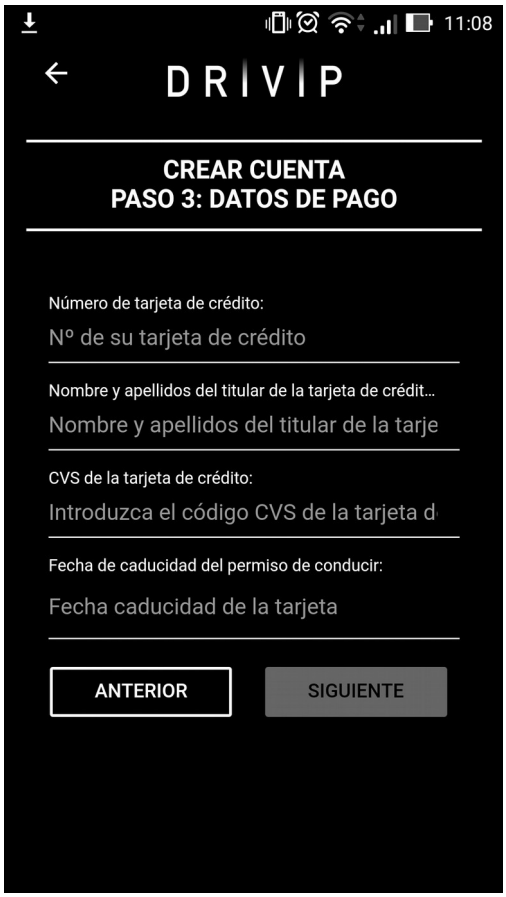


Ilustración 42: Paso 3/4 registro

Ilustración 43: Paso 4/4 registro

Ahora que ya estamos registrado si que podremos reservar y alquilar vehículos. Como se ve en la figura 44 si reservamos un vehículo esté se mostrará en negro.

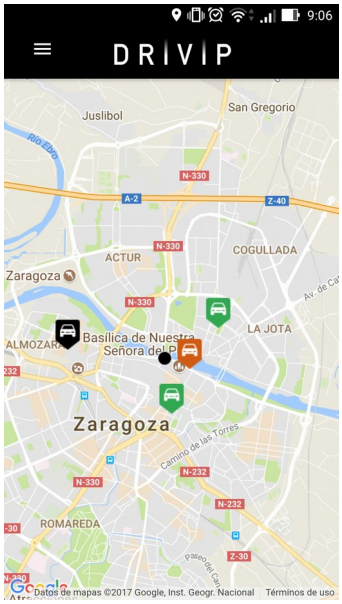


Ilustración 44: Coche reservado.

Si tenemos un coche reservado no se nos permitirá reservar otro o alquilar un coche distinto. Si queremos reservar/alquilar otro coche tendremos que liberar previamente el coche que tenemos reservado. Cuando alquilamos un coche se muestra un mensaje de que se han abierto las puertas del vehículo y ha comenzado el alquiler. A continuación solo se nos mostrará el coche que tenemos alquilado. Figura 45.

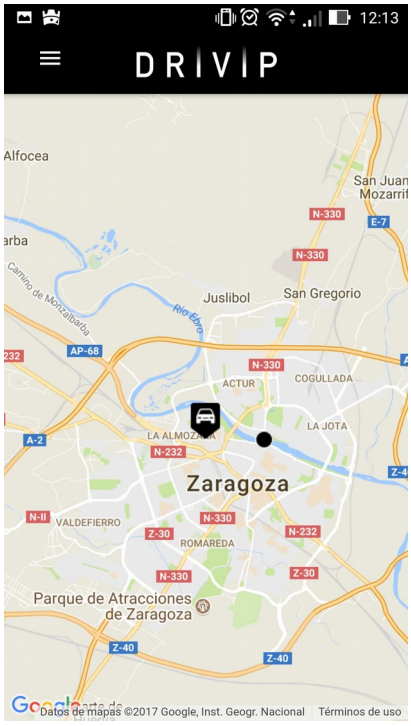


Ilustración 45: Coche alquilado.

Si seleccionamos nuestro coche se nos muestra información sobre el alquiler y la opción de terminar el alquiler.
Al finalizar el viaje se nos informará del tiempo de alquiler y del coste de este. Figura 46.

Ilustración 46: Información sobre el alquiler.

Para acceder al perfil tendremos que ir al menú y seleccionar mi perfil.
En esta página se nos muestra un resumen de los datos del usuario:

- Numero de viajes
- Dinero total gastado.
- Tiempo total de los viajes
- Media de coste por viaje

Además a continuación se nos muestra un listado de todos nuestros viajes y si seleccionamos uno se ampliará la información de este. Figuras 47 y 48.

Ilustración 47: Mi perfil

Ilustración 48: Mi perfil con detalles de un viaje.

Por ultimo tenemos la página de login, a la cual accederemos desde el menu y seleccionado iniciar sesión. Tendremos que introducir nuestro correo y contraseña para iniciar sesión, o, seleccionar la opción registrarse si es el caso. Figura 49.

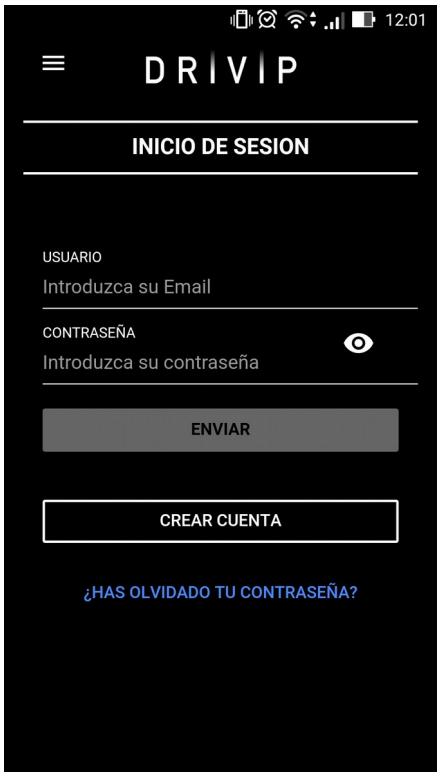


Ilustración 49: Inicio de sesión.

Bibliografía

1: Neodoo microsystems, Neodoo, , http://neodoo.es/
2: OSvehicle, Osvehicle, , https://www.osvehicle.com/
3: RapsBerry, RapsBerry, , https://www.raspberrypi.org/
4: Tesla, Tesla, , https://www.tesla.com/es_ES/
5: APiari, Documentacion API Tesla, , https://timdorr.docs.apiary.io/#
6: RESTEasy, RESTEasy, , http://resteasy.jboss.org/
7: MQTT, Protocolo MQTT, , http://mqtt.org/
8: Ionic 2, Ionic 2, , https://ionicframework.com/docs/
9: OAuth, OAuth, , https://oauth.net/2/
10: MariaDB, MariaDB, , https://mariadb.org/
11: Eclipse IDE, Eclipse, , https://www.eclipse.org/
12: Maven, Maven, , https://maven.apache.org/
13: Git, Git, , https://git-scm.com/
14: GitHub, GitHub, , https://github.com/
15: CircleCi, CircleCi, , https://circleci.com/
16: HTTPS, HTTPS, ,
https://es.wikipedia.org/wiki/Protocolo_seguro_de_transferencia_de_hipertexto
17: RESTEasy, RESTEasy, , http://resteasy.jboss.org/
18: Servlet, Servlet, , https://es.wikipedia.org/wiki/Java_Servlet
19: Apache Tomcat, Apache Tomcat, , http://tomcat.apache.org/
20: JSON, JSON, , https://www.json.org/json-es.html
21: Jackson, Jackson, , https://www.mkyong.com/java/jackson-2-convert-java-object-to-from-json/
22: Hibernate, Hibernate, , http://hibernate.org/
23: JPA, JPA, , https://es.wikipedia.org/wiki/Java_Persistence_API
24: Mosquitto, Mosquitto, , https://mosquitto.org/
25: JUnit, JUnit, , http://junit.org/junit5/
26: Apache Cordova, Apache Cordova, , https://cordova.apache.org/
27: Sass, Sass, , http://sass-lang.com/
28: AngularJS, AngularJS, , https://angularjs.org/
29: Virtualbox, Virtualbox, , https://www.virtualbox.org/
30: Vagrant, Vagrant, , https://www.vagrantup.com/
31: Ansible, Ansible, , https://www.ansible.com/
32: Postman, Postman, , https://www.getpostman.com/